Mathias Weller

Aspects of Preprocessing Applied to Combinatorial Graph Problems

Mathias Weller

Aspects of Preprocessing Applied to Combinatorial Graph Problems

Mathias Weller

Aspects of Preprocessing Applied to Combinatorial Graph Problems

Universitätsverlag der TU Berlin

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.dnb.de/ abrufbar.

Universitätsverlag der TU Berlin 2013
http://www.univerlag.tu-berlin.de

Fasanenstr. 88 (im VOLKSWAGEN-Haus), 10623 Berlin
Tel.: +49 (0)30 314 76131 / Fax: -76133
E-Mail: publikationen@ub.tu-berlin.de

Zugl.: Berlin, Technische Universität, Diss., 2012
1. Gutachter: Prof. Dr. Rolf Niedermeier,
2. Gutachter: Prof. Dr. Gerhard Woeginger
3. Gutachter: Prof. Dr. Christophe Paul
Die Arbeit wurde am 5. Dezember 2012 unter Vorsitz
von Prof. Dr. Uwe Nestmann erfolgreich verteidigt.

Druck: docupoint GmbH, Magdeburg
Satz/Layout: Mathias Weller
Umschlagfoto: Daniel Gilbey | Dreamstime Stock Photos

Zugleich online veröffentlicht auf dem Digitalen Repositorium
der Technischen Universität Berlin:
URL http://opus.kobv.de/tuberlin/volltexte/2013/3878/
URN urn:nbn:de:kobv:83-opus-38783
http://nbn-resolving.de/urn:nbn:de:kobv:83-opus-38783

ISBN 978-3-7983-2508-1 (Druckversion)
ISBN 978-3-7983-2509-8 (Onlineversion)

Zusammenfassung

In dieser Arbeit werden verschiedene Aspekte effizienter (in polynomieller Zeit ausführbarer) Vorverarbeitung für NP-schwere Probleme untersucht. Besonderer Schwerpunkt liegt dabei auf praktisch relevanten Eigenschaften der Vorverarbeitung. Zur Untersuchung dieser Eigenschaften dient die parametrisierte Komplexitätstheorie, da sie die nötigen Werkzeuge zur Analyse von Vorverarbeitungsalgorithmen bereitstellt. In der parametrisierten Komplexitätstheorie wird die Komplexität eines Problems nicht nur in der Größe der Eingabe, sondern auch im „Parameter“, einer natürlichen Zahl, die eine Eigenschaft der Eingabe repräsentiert, gemessen. Dadurch lässt sich die Güte einer Vorverarbeitung, die eine zur Eingabe äquivalente, jedoch kleinere Instanz produziert beschreiben, indem die Ausgabe des Vorverarbeitungsalgorithmus durch eine Funktion im Parameter nach oben beschränkt wird. Hierbei gilt üblicherweise das Motto „je kleiner desto besser“. Ein solcher Algorithmus wird auch als *Kernelisierung* bezeichnet, und die Ausgabe einer Kernelisierung ist ein *Problemkern*.
Die folgenden Aspekte effizienter Vorverarbeitung werden in der Arbeit diskutiert.

Nichtstandardparameter. In der Vergangenheit war es üblich die Größe der geforderten Lösung als Parameter zu wählen, um das vorliegende Problem zu untersuchen. Allerdings ist es meist sinnvoll, andere Parameter zu betrachten. Im Kontext von Graphproblemen bieten sich hier Eigenschaften des gegebenen Graphen an. Prominente Beispiele sind die Baumweite, die Größe einer kleinsten Knotenüberdeckungsmenge oder die Kantenlöschungsdistanz zu Bäumen (die Anzahl Kanten, die mindestens aus dem Graphen entfernt werden müssen, damit dieser ein Baum wird). Letzteres ist eine untere Schranke für die Kantenlöschungsdistanz zu Raupen („Caterpillar trees“). Diese Distanz zu berechnen ist das Ziel des NP-vollständigen Problems Two-Layer Planarization, das im Bereich des Zeichnens von Graphen interessant ist.
Two-Layer Planarization wird bezüglich des Parameters „Kantenlöschungsdistanz f zu Bäumen“ betrachtet, wobei ein Problemkern der Größe $O(f)$, der in „fast linearer Zeit“ berechnet werden kann, gezeigt wird. Weiterhin lässt sich das Problem mit einem Algorithmus, der auf den Problemkern aufbaut, in $O(3.8^f \cdot f^2 + f \cdot |G|)$ Schritten lösen. Dieser Algorithmus wird mit bisherigen Lösungsalgorithmen, die für den Standardparameter entwickelt wurden [182], verglichen.

Effiziente Vorverarbeitung. Im Rahmen exakter Algorithmen für NP-schwere Probleme ist „effizient“ üblicherweise synonym zu „in Polynomzeit berechenbar“. Allerdings erfordern es manche praktische Anwendungen, dass ein Datensatz in

Linearzeit verarbeitet wird. Daher ist es interessant, welchen Grad der Eingabevereinfachung man in linearer Zeit erreichen kann. Daher soll die Aufmerksamkeit auf Vorverarbeitungen gelenkt werden, die in Linearzeit berechenbar sind. Solche Vorverarbeitungen sind besonders in Verbindung mit Approximationen, die in Linearzeit berechnet werden können interessant, lassen sich allerdings auch mit anderen Algorithmen kombinieren. Insbesondere lassen sich „schnelle“ Kernelisierungen mit „effektiven“ kombinieren, sodass sich kleine Problemkerne schneller berechnen lassen.
Eine Kernelisierung für das Dominating Set Problem auf planaren Graphen bezüglich des Parameters „Größe γ einer kleinsten dominierenden Menge[1] im Eingabegraphen“ wird entworfen (das Dominating Set Problem fragt, ob der Eingabegraph eine dominierende Menge einer gegebenen Größe enthält). Der entsprechende Problemkern basiert auf Datenreduktionsregeln von Alber et al. [8], lässt sich allerdings in Linearzeit berechnen und hat überdies noch lineare Größe in γ.

Vorverarbeitung über Kernelisierung hinaus. Besonders in der parametrisierten Komplexitätstheorie wird der Begriff „Vorverarbeitung“ meist mit „Kernelisierung“ gleichgesetzt. Am Beispiel des bekannten Rural Postman Problems wird gezeigt, dass dies nicht immer der Fall sein muss. Häufig lassen sich heuristische Vorverarbeitungen oder Vereinfachungen der Eingabe die nicht deren Größe betreffen finden, die die Eingabe deutlich vereinfachen können. Rural Postman lässt sich interpretieren als Weighted Multigraph Eulerian Extension, in dem es gilt, eine gegebene Zahl von Kanten in den gegebenen (gerichteten und kantengewichteten) Multigraphen einzufügen, sodass der resultierende Multigraph Eulersch ist. Eine Algorithmus wird entwickelt, der auf Dynamischem Programmieren basiert, und Instanzen mit n Knoten in $O((2n)^k \cdot n^4)$ Schritten lösen kann (wobei k die kleinste Größe einer Eulerschen Erweiterung der Eingabe bezeichnet, die ein gegebenes Maximalgewicht nicht überschreitet). Durch eine intuitive Vorverarbeitung erlaubt die Berechnung einer Lösung in $O(4^k \cdot n^3)$ Schritten. Diese Vorverarbeitung stellt bemerkenswerterweise keine Kernelisierung dar; In der Tat lässt sich zeigen, dass es bezüglich dieses Parameters keinen Problemkern polynomieller Größe gibt, sofern es nicht zu einem (weitgehend für unwahrscheinlich gehaltenen) Kollaps der Polynomialzeithierarchie kommt.

Zwischen Turing- und klassischer Kernelisierung. Die meisten der in der Vergangenheit gezeigten unteren Schranken für die Größe von Problemkernen basieren auf „Kompositionsalgorithmen“ [26, 95]. Gibt es für ein Problem einen

[1] Eine dominierende Menge ist eine Knotenmenge deren vereinigte (geschlossene) Nachbarschaft alle Knoten des Graphen enthält.

solchen Kompositionsalgorithmus, so sind Problemkerne polynomieller Größe wahrscheinlich nicht in Polynomzeit berechenbar. In einem solchen Falle ist es sinnvoll den Kernelisierungsbegriff zu erweitern. Verallgemeinerungen auf die das kompositionsbasierte Werkzeug nicht anwendbar ist, die aber dennoch stark genug sind um praktische Relevanz zu haben sind die sogenannten „Turing-Kernelisierungen“, die die Möglichkeit haben, mehrere größenbeschränkte Instanzen zu erzeugen (gegenüber einer einzigen Instanz bei klassischer Kernelisierung). Hier wird eine spezielle Form der Turing-Kernelisierung betrachtet, die sich durch exzellente Parallelisierbarkeit auszeichnet, eine Eigenschaft, die in der Praxis sehr interessant ist.
Verschiedene Möglichkeiten, Turing-Kernelisierungen in gewissen Aspekten auf Kosten anderer Aspekte zu verbessern, werden aufgezeigt. Zum Beispiel lässt sich die beschriebene Parallelisierbarkeit „eintauschen“, um die Zahl der erzeugten Instanzen zu verringern. Am Beispiel des Problems Clique (das nach einem vollständig verbundenen Teilgraphen einer gegebenen Größe fragt) wird demonstriert, wie sich die Größe der einzelnen erzeugten Instanzen durch Erzeugen zusätzlicher Instanzen reduzieren lässt.

Abstract

This work considers multiple aspects of efficient (that is, polynomial-time executable) preprocessing for NP-hard problems with emphasis on practically relevant properties. A theoretical framework for the development and analysis of preprocessing algorithms is supplied by parameterized complexity theory. Herein, the complexity of a problem is measured not only in the size of the input, but also in some other aspect measurable by a natural number. This aspect is called the "parameter". Then, a *kernelization* is a polynomial-time algorithm that, given an instance, computes an equivalent, smaller instance (the *problem kernel*) whose size can be bounded by a function of the parameter. This allows measuring the performance of a preprocessing (that is, the size of the output instance). Here, the usual motto is "the smaller the output instance, the better the preprocessing algorithm". In particular, we consider the following aspects of preprocessing.

Non-standard parameters. In the past, the most frequently considered parameter was the size of the sought solution. However, it often makes sense to select a different parameter. Especially in the context of graph problems, the structure of the given graph is a rich source of parameters. Prominent examples are the treewidth, the size of a smallest vertex cover, or the feedback edge set number (the smallest number of edges whose removal destroys all cycles in the graph). The latter is a lower-bound for the *biplanarization number*, that is, the minimum number of edges whose removal turns the graph into a forest of so-called "caterpillar trees". To check whether a graph has a certain biplanarization number requires solving the NP-complete Two-Layer Planarization problem, which has applications in graph drawing.
We consider the parameter "feedback edge set number f" for Two-Layer Planarization and show that a problem kernel of size $O(f)$ can be computed in "almost linear time". We also develop an algorithm that solves the problem in $O(3.8^f \cdot f^2 + f|G|)$ time and compare its implementation to an implementation of a previous algorithm by Suderman [182] for Two-Layer Planarization.

Efficient preprocessing. In the context of parameterized complexity, "efficient" is usually equivalent to "computable in polynomial time". Some high-throughput real-wold applications, however, may have stricter requirements regarding the running time of the preprocessing algorithms. In this case, a linear-time preprocessing algorithm would be more practical. Thus, we want to shift focus to the running times of kernelization procedures. This is especially interesting when combining kernelizations with approximation algorithms or other kernelizations.

This way, a "fast" kernelization can be combined with an "effective" kernelization to speed up computation of a small problem kernel.
Here, we consider the Dominating Set problem on planar graphs with respect to the parameter "size γ of a minimum dominating set[2] in the input graph". The Dominating Set problem asks whether there is a dominating set of a given size in the input graph. Based on previous work by Alber et al. [8], we develop a kernelization that runs in linear time and outputs a problem kernel of size $O(\gamma)$.

Preprocessing beyond kernelization. In parameterized complexity, the term "preprocessing" is often considered a synonym of "kernelization". However, heuristic data reduction or other simplification can speed up the computation of an optimal solution. We use the well-known Rural Postman problem, which has numerous applications in routing, to show how preprocessing that does not constitute a kernelization helps simplify instances so they can be solved provably faster.
We interpret Rural Postman as Weighted Multigraph Eulerian Extension, which is the problem of turning a given arc-weighted directed multigraph Eulerian by adding a given number of arcs. We develop a dynamic-programming based algorithm that solves Weighted Multigraph Eulerian Extension in $O((2n)^k \cdot n^4)$ time, where k denotes the smallest number of arcs to add to the input multigraph such that the result is Eulerian and the total weight of additional arcs does not exceed a given bound. We provide an intuitive preprocessing algorithm that allows computing an optimal solution in $O(4^k \cdot n^3)$ time. This preprocessing does not constitute a kernelization with respect to k. Moreover, we even show that, unless an unexpected collapse of the polynomial hierarchy occurs, no polynomial-time preprocessing can construct a polynomial-size problem kernel for Weighted Multigraph Eulerian Extension.

Between Turing and classical kernelization. Most of the lower bounds for the size of problem kernels are based of composition algorithms [26, 95]. If a parameterized problem is unlikely to admit a polynomial-size kernel, then a relaxation of the requirements for kernelizations may be helpful to formulate a preprocessing for the problem at hand. An example for such a relaxation is the "Turing kernelization", which allows creating multiple instances instead of just one. It can be argued that this type of kernelization is still practically relevant. In this thesis, we consider a form of Turing kernelization that lies between Turing and classical kernelization. This "Truthtable kernelization" is excellently suited for massive parallelism, which today's computation environments clearly support.

[2] A dominating set in a graph is set of vertices whose united (closed) neighborhood contains all vertices of the graph.

We present several ways of "trading off" properties of Truthtable kernelizations. For example, the number of created instances may be reduced by giving up the described parallelizability. Furthermore, we use the Clique problem (which asks whether the input graph has a clique of given size) as an example to show how the size of each created instance can be reduced at the cost of creating more instances.

Preface

This thesis summarizes large parts of my work of the past years, in which I was supported by the Deutsche Forschungsgesellschaft (DFG) under the project name "Datenreduktion und Problemkerne" (DARE, project numbers GU 1023-1/1 and NI 369-11/2).

The thesis is structured into six chapters, representing parts of my work in the field of parameterized complexity theory. Chapter 1 contains an introduction to and motivation of preprocessing, as well as preliminary explanations of graph-theoretic and (parameterized) complexity-theoretic notation used throughout the thesis.

Chapter 2 contains work I did in collaboration with my coauthor Johannes Uhlmann on the Two-Layer Planarization problem [191] which aims at making a given graph drawable in two layers without edge crossings by deleting edges. As we shared a room in our offices at the Friedrich-Schiller-Universität Jena, he approached me with the suggestion to look at this problem. Based on earlier work he did with Nadja Betzler and Jiong Guo, there was a stub manuscript containing some data reduction rules. In joint work, Johannes and I developed missing data reduction rules, I came up with a uniform way of presenting our preprocessing using "tokens". We proved correctness of the whole procedure and developed a branching strategy to solve Two-Layer Planarization. The paper then was accepted to publication at the *7th Annual Conference on Theory and Applications of Models of Computation (TAMC 2010)*, yielding an invitation to a special issue of the journal *Theoretical Computer Science*. Recently, I picked the paper up again and worked on the question whether the branching algorithm presented by Suderman [182] could be adapted to our parameterization. I succeeded to some extend, providing a branching algorithm whose asymptotic running time almost matches that of Suderman's algorithm. I implemented the algorithm and tested it against published results by Mutzel [156] and Suderman and Whitesides [183]. This work, however, was not published so far.

Chapter 3 considers the Planar Dominating Set problem, which asks for a set of vertices that dominate all other vertices in a planar graph. It contains a linear-time variant of the known linear-size kernel for the parameter "solution size". Rolf Niedermeier introduced the idea of emphasizing the running time of kernelization algorithms, since they are inherently combinable, achieving fast running times and small problem kernel sizes as a result. Much of the work was done when Frank Kammer visited our group at TU Berlin. There have been countless debates, suggestions, bugfixes and reworks involving all authors of the resulting paper: René van Bevern, Sepp Hartung, Frank Kammer,

Rolf Niedermeier, and myself. The final result [19] was presented at the *6th International Symposium on Parameterized and Exact Computation (IPEC 2011)*. At the same conference, Torben Hagerup presented a similar theorem, providing a different kernelization approach [109]. Whereas we improved the running time of a previous algorithm [8], he designed a completely new algorithm. Although he stated that his approach was much simpler than ours, we thought that, in light of the application of the kernelization of Alber et al. [8] to other combinatorial problems on planar graphs [194], our approach is more versatile and it is merely our proof of correctness that is highly complex, not the algorithm that we give.

Chapter 4 discusses the Eulerian Extension problem, which was brought to my attention by Jiong Guo, who attended one of Wiebke Höhn's talks about work she did at TU Berlin [119] regarding Eulerian extension problems. Most work was done while Frederic Dorn visited our group at the Friedrich-Schiller-Universität Jena. In particular, my coauthors (Frederic Dorn, Hannes Moser and Rolf Niedermeier) and I found that a version of the problem called Weighted Multigraph Eulerian Extension was basically equal to Rural Postman, a fundamental problem in arc routing that is closely related to the well known Chinese Postman problem. I worked out the details of polynomial-time solvable special cases and, together with Hannes Moser, developed the idea of using the Black/Gray Weighted Multigraph Eulerian Extension problem to solve Weighted Multigraph Eulerian Extension. I presented the resulting paper at the *36th International Workshop on Graph Theoretic Concepts in Computer Science (WG 2010)* [65]. A revised version of the long version is currently in production at the *SIAM Journal on Discrete Mathematics*.

In Jena, the student Manuel Sorge approached our group to supervise his diploma thesis. He accepted to work on the question whether Weighted Multigraph Eulerian Extension was fixed-parameter tractable with respect to the parameter "number of connected components in the input graph" that we identified as a major challenge (based on statements of Orloff [164] and Frederickson [97]) in this context. The results yielded two papers that were presented at the the *22nd International Workshop on Combinatorial Algorithms (IWOCA 2011)* [175] and the *37th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2011)* [176]. A long version combining them appeared in the *Journal of Discrete Algorithms* [178]. Of the results in this work, Chapter 4 contains a sketch of the incompressibility proof for Weighted Multigraph Eulerian Extension with respect to the parameters "sum of imbalances", "number of connected components" and "number of arcs in a solution".

Since recently, we have Georg Hieronimus, a student of the TU Berlin, implement our algorithms [65, 178] and we have initiated a cooperation with the Berliner Stadtreinigung, who organizes the local snow plowing service for Berlin.

The presence of our research in the field of arc routing led to Ángel Corberán and Gilbert Laporte inviting us to write a chapter on the complexity of arc routing problems, which is to be published by SIAM [20].
Chapter 5 contains yet unpublished work of Rolf Niedermeier and me, describing new directions of kernelization with an emphasis on variants of Turing kernelization, which is a form of preprocessing that allows more than one instance to be created. The idea to examine variants of the rather young concept of Turing kernelization originated from the observation that no known Turing kernel makes use of the power this concept provides. Limiting this power can, on the other hand, lead to more general observations regarding this kind of preprocessing. Since Turing kernelization does not seem significantly less practical than classical (many one) kernelization, we deemed researching the possibilities of this kind of preprocessing interesting. Pointed to the notion of truthtable reducibility by Rolf Niedermeier, I formulated the parallel concept of truthtable kernelization and showed an analogon to Beigel's theorem for truthtable and Turing reductions [15]. I started searching for a problem to which this meta-theorem was applicable, but had no success yet. I came up with the parameter "subgraph cutset number" and constructed truthtable kernels for Clique with respect to this parameter whose size and number of queries can be traded for one another continuously.
Chapter 6 concludes this thesis with a resumé of ideas and results presented in this work. This chapter also contains a compilation of open problems and a preview on possible future research topics and concepts.

Acknowledgments

My interest in theoretical computer science was first piqued by a course about discrete mathematics that I attended at the Friedrich-Schiller-Universität Jena. For giving that course in the way he did, I want to thank Jörg Vogel.
I am deeply indebted to my supervisor Rolf Niedermeier, who also supervised my diploma thesis. He guided me safely through my scientific life to this very day. It is amazing how innovative a man of his experience can be and he somehow manages to always ask the right questions.
I want to thank my colleagues and former colleagues Nadja Betzler, René van Bevern, Robert Bredereck, Jiehua Chen, Michael Dom, Jiong Guo (now at the Cluster of Excellence Multimodal Computing and Interaction, Saarbrücken), Sepp Hartung, Falk Hüffner, Christian Komusiewicz, Hannes Moser (now with Geomagic GmbH, Leipzig), André Nichterlein, Manuel Sorge, Ondřej Suchý (now at Czech Technical University, Prague, Czech Republic) and Johannes Uhlmann, who made work pleasent and enjoyable and often provided interesting and fun conversation at lunchtime.
My coauthors outside the research group surrounding Rolf Niedermeier shall also be mentioned here. Thanks for the fruitful discussions and other contributions to our joint works. Their names are Frederic Dorn (SINTEF Energy Research, Trondheim, Norway), Martin Dörnfelder (IDIADA Fahrzeugtechnik, München), Rudolf Fleischer (German University of Technology in Oman, Maskat, Oman), Frank Kammer (Universität Augsburg), Matthias Mnich (Cluster of Excellence Multimodal Computing and Interaction, Saarbrücken), Yihui Wang (Fudan University, Shanghai, China), and Xi Wu (University of Wisconsin-Madison, Madison, USA).
It is with great urge that I want to thank everyone who supported me throughout my studies, in particular my family who was and will always be there for me, and the people that I am proud to call my friends (you know who you are)!
Finally, I want to thank Christophe Paul and Gerhard Woeginger for taking the time to review this thesis.

Contents

Chapter 1

Introduction & Preliminaries

1.1 Introduction

Preprocessing, or rather precomputation, can be characterized as the precautionary measure of spending time once to save time (indefinitely) many times later. It is an ancient tool to speed up mathematical computations. While in many ancient civilizations, farmers would, for example, use multiplication tables to compute the size of an acre, tables of logarithms, square roots, and triangular functions survived until the dawn of electronic computation. Probably one of the most impressive examples of ancient precomputation are the Babylonian lists found at Senkerah on the Euphrates in 1854 [163]. These 4000-year old lists contained the squares of natural numbers up to 59. The Babylonians used these lists to multiply supported by the formula

$$x \cdot y = \frac{(x+y)^2 - x^2 - y^2}{2}.$$

While it is arguable whether this is indeed an efficient way to multiply, it undoubtedly is evidence for preprocessing being considered a powerful tool in mathematics and algorithmics for as long as mathematics has been cultivated by humanity.
In modern days, as algorithmic theory and mathematical analysis thereof developed, preprocessing mainly lived as the idea to store precomputed solutions to subtasks that would be done repeatedly in the main algorithm to solve some problem. Naturally, the effectiveness of the preprocessing grows with the number of times a subtask is repeated. The following examples form only the tip of the iceberg of practical applications of efficient preprocessing; more can be found in the numerous survey articles [23, 105, 145].

- A cardplayer sorting his hand performs a preprocessing in order to find desired cards more quickly. In data structures, storing an array A of integers in sorted order allows for queries of the form "does A contain x?" to be answered in logarithmic time, instead of linear time in the size of A.

- In asymmetric cryptography, more specifically, in Garner's algorithm [170, 154], precomputing certain remainders allows the use of the Chinese remainder theorem to speed up deciphering an encoded message.
- For the Deutsche Bahn A.G., Weihe [196, 197] considered the Path Cover problem which, given a graph G, a set $\mathcal{P}$ of paths in G and an integer k, asks whether there is a vertex set V' of size at most k such that each path in $\mathcal{P}$ contains a vertex of V'? Weihe [196, 197] developed a data reduction that shrunk the massive train networks to some rather small components that could easily be solved by brute force.
- In Machine Learning, preprocessing plays an important role as it is used for data cleaning, noise reduction, dimension shrinking, and accounting for time-sequence information [111, 98].
- Modern industry-strength integer linear programming solvers highly depend on preprocessing to reduce redundancies and simplify constraints [1]. The ILOG CPLEX solver, for instance, knows two kinds of preprocessing: *primal reductions* are independent of the objective function and *dual reductions* are independent of the right-hand side of the constraining inequalities. An impressive example of the power of preprocessing in solving integer linear programs was presented by Bixby [22].

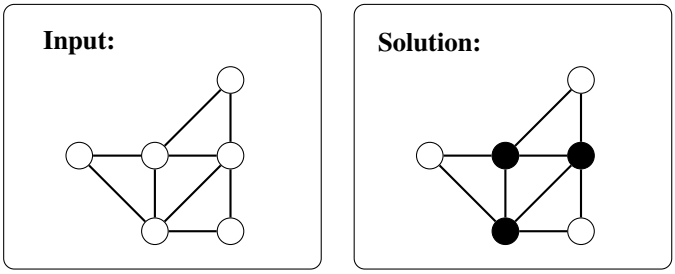
Polynomial-time preprocessing is very versatile in that it fits well into many approaches of solving problems. In parameterized algorithmics, polynomial-time preprocessing is used to obtain "kernels" (see Section 1.2.4) that speed up exact algorithms for NP-hard problems. Many such kernels can also be combined with approximation algorithms instead. Harnik and Naor [113] use "compression", which is very much alike kernelization, to store instances efficiently for later processing and to compute one-way functions from sampleable distributions, which are useful in cryptography.

Considering the power that preprocessing exhibits, it is natural to try and harness this power to speed up solutions for computational problems that are notoriously hard, but still occur often in practice (we are of course referring to NP-hard problems). However, a concise theory of preprocessing using mathematical analysis eluded these problems until the dawn of parameterized complexity. This is because, in classical complexity theory, the complexity of a problem is measured as the number of computation steps for an input of size n. If now some efficient (that is, polynomial-time computable) preprocessing algorithm could simplify, that is, shorten an input provably, then repeated application of this preprocessing algorithm would solve the problem in polynomial-time. However, since NP-hard problems are unlikely to be solvable in polynomial time, the existence of such preprocessing algorithms is also unlikely. Therefore, this thesis works within the

framework of parameterized complexity, where the complexity of a problem is measured not only in the length of the instances but rather an additional aspect, called the "parameter", of the instance. This aspect could be anything measurable by a natural number. Clearly, the choice of parameter can be good or bad depending on the application of the problem. For instance, consider the following problem: A set of scientific experiments have been conducted but some outcomes contradict each other. Then, we would have to choose a set of experiments to repeat or ignore in order to develop a conflict-free theory. Then, Occam's razor suggests that ignoring the smallest set of experiments leads to the most promising conflict-free theory. The (graph-theoretic formulation of the) problem of finding such a set is called Vertex Cover. Now, we may assume that the experiments have been conducted cautiously, giving us hope that the number of experiments we have to ignore is small. Hence, this number would constitute a good parameter. In fact, it turns out that Vertex Cover can be solved in polynomial time if the number of ignored experiments is at most logarithmic in the number of experiments that were carried out [47].

In this thesis we mostly consider NP-complete graph problems such as the Vertex Cover problem described earlier. We repeat it here in its graph-theoretic formulation, which also gives us the opportunity of familiarizing the reader with our presentation of problems. We introduce problems as a description of the input and a question that characterizes the task. Usually, we then give a small simplified example of what a possible input and solution of the problem looks like.[3]

Vertex Cover
Input: An undirected graph G and an integer k.
Question: Is there a vertex set $V' \subseteq V(G)$ such that $|V'| \leq k$ and removing all vertices in V' from G results in an edgeless graph?



[3] Keep in mind that, formally, it is not necessary to compute a solution for the decision problem since we just have to decide whether the given instance is a yes- or no-instance.

If appropriate (for example, for less important problems), we may omit or marginally simplify (by omitting weights, for example) the illustrating figure to maintain a clean layout. For parameterized problems, we augment the definition by giving the parameter for the problem.
Graph problems are incredibly versatile, that is, they can be used to model a wide range of problems occurring in real world. Furthermore, polynomial-time algorithms are known for a large bundle of common tasks regarding graphs. This comes in handy when developing efficient preprocessing algorithms. Finally, graphs have a large number of measurable properties that are candidates for parameterization and can be compared, effectively giving rise to a "map" of parameteres that can be "navigated" in order to find a suitable parameterization for a given problem.

1.1.1 Outline

In this thesis, we consider different aspects of preprocessing. After agreeing on notational basics in Section 1.2, we present exemplary work on four directions of research in the context of polynomial-time preprocessing: Kernelization for non-standard parameters, preprocessing beyond kernelization, running time in kernelization, and extensions of the kernelization concept.

Kernelization for Non-Standard Parameters. Graph problems are mostly stated as "given a graph and some integer k, does something of size k exist?". In the early days of parameterized complexity, it was considered standard to choose the "solution size", meaning the integer k, as parameter for a problem. As sketched in Section 1.1, this seems natural for minimization problems as Vertex Cover since very large solution sets are often useless in the practical application. However, the following arguments show that it can be preferable to consider parameters that capture the "complexity" of the input graph. These parameters are called *structural parameters*.

- The solution size is not always small, especially for maximization problems. This is maybe best exhibited by Maximum Satisfyability which, given a boolean formula in conjunctive normal form, asks for an assignment of the variables that satisfies at least a given number of clauses. Here, the standard parameter "number of clauses to satisfy" is large for non-trivial instances [146]. This is because an assignment satisfying at least half of the clauses always exists.
- Choosing the solution size as parameter completely ignores the structure of the input graph. However, this is usually a great way of limiting the set of inputs that can be expected to be seen in practice. For example, consider

problems on graphs representing road maps. They can be expected to be close to planar graphs and, thus, a distance measure to planarity would be a promising parameterization. In this way, it can be argued that the structure of the graph provides the most promising parameters.

- Considering structural parameters opens up a whole space of parameters that can be navigated using the "stronger" relation. For example, if the problem at hand proves to be hard for a certain parameter, we may, instead of giving up, consider another parameter that is "weaker" than the first. Here, "is weaker" means "can be lower-bounded by".[4] Practical examples include the Two-Layer Planarization problem (see Chapter 2), where the feedback edge set of the input is a lower bound for the solution size, and 2-Union Independent Set that is $W[1]$-hard for the standard parameter [129] but allows polynomial kernels for several "weaker", structural parameters [21].

In Chapter 2, we demonstrate how non-standard structural parameters can help design practically relevant preprocessing. The Two-Layer Planarization problem, which originates in the field of graph drawing, serves as an example for this demonstration.

Running Time in Kernelization. A popular concept in parameterized complexity is the "race" for better results [3, 134]. Notably, there are two kinds of races in the community. For the most important problems and their most popular parameters, there is the race for the fastest algorithm solving the problem and the race for the smallest kernel. However, an important aspect of preprocessing seems to have gotten lost: the running time of the preprocessing algorithm. Especially in light of "chaining" preprocessing algorithms, that is, first running the fast kernelization, then running the slower, more powerful kernelization, practical algorithms can benefit greatly from improving the running times of known preprocessing algorithms. In Chapter 3, we emphasize the importance of this aspect and develop a linear-time variant of the celebrated linear-size kernelization for Planar Dominating Set [8].

Preprocessing Beyond Kernelization. With the recent rise of parameterized complexity and especially kernelization, efficient preprocessing developed into a synonym for kernelization. However, polynomial-time preprocessing helps solve problems also beyond kernelization. We demonstrate this in Chapter 4 by developing a dynamic programming algorithm for the Eulerian Extension problem that profits enormously from preprocessing. Notably, this preprocessing does not constitute a kernelization. In fact, we briefly sketch a proof showing

[4] Parts of this relation for a selected range of structural parameters can be found in Section 1.2.4.

that a polynomial-size kernel with respect to the parameter we are considering is unlikely to exist.

Between Turing and Classical Kernelization. With the development of lower-bound techniques for kernelization [26, 29, 58, 95, 116, 136, 155] came the need for less strict concepts of preprocessing. Although multiple attempts have been made to overcome this "kernelization hardness" [17, 85, 88, 127, 128, 143], the most widely accepted concept of preprocessing with performance guarantees for problems "without polynomial kernels" is the Turing kernelization [88, 143]. Turing kernelization generalizes classical kernelization by allowing the creation of multiple instances, such that answers to each of these instances allow solving the original instance in polynomial time. However, this relatively general concept may not be best-suited in some practical situations. To date, no known Turing kernelization makes use of the full strength of the concept. Therefore, it is interesting to develop a slightly less general concept that still catches all previous results and allows global statements and theorems that do not hold for the currently used concept of Turing kernelization. We focus on a kernelization variant between classical and Turing kernelization and explore some of these concepts in Chapter 5, proving some general statements for these kernelization variants.

1.2 Preliminaries

In this section, we introduce basic notation used throughout this thesis. First, we negotiate graph-theoretic concepts, then briefly introduce computational complexity, NP-hardness, and parameterized complexity theory, including the central concept of this thesis: kernelization.

1.2.1 Graph Theory

Undirected Graphs. An undirected graph (or just plain "graph") is a finite construct of objects, called "vertices" (usually drawn as circles or dots) and connections between vertices, called "edges" (usually drawn as lines between vertices). Each edge is represented by an unordered pair of vertices. In this sense, a *graph* is an ordered pair containing the sets V (vertices) and $E \subseteq \{\{u, v\} : u, v \in V\}$. We write $\binom{V}{2}$ as a shorthand for the set of all unordered pairs of vertices. Throughout this thesis, we use $n := |V|$ and $m := |E|$.
In the following, let G be a graph. We denote the set of all vertices of G by $V(G)$ and the set of all edges of G by $E(G)$. We abbreviate $|V| + |E|$ to $|G|$. A graph G' that can be obtained by deleting vertices and edges from G is called a *subgraph*

of G. For every vertex set $V' \subseteq V(G)$ and vertex $v \in V(G)$, we denote the subgraph of G that is *induced by* V' by $G[V'] := (V', E(G) \cap \{\{u, v\} : u, v \in V'\}$. To allow easy parsing of set exclusions, we use the operators "$-$" for vertex sets V' and vertices v, and "$\dot{-}$" for edge sets E' and edges e as follows.

$$V' - v := V' \setminus \{v\} \qquad E' \dot{-} e := E' \setminus \{e\}$$
$$G - V' := G[V(G) \setminus V'] \qquad G \dot{-} E' := (V(G), E(G) \setminus E')$$
$$G - v := G - \{v\} \qquad G \dot{-} e := G \dot{-} \{e\}.$$

Let $V(E') := \bigcup_{e \in E'} e$ denote the set of endpoints of edges in E'. The *complement* of G is $\overline{G} := (V(G), \binom{V}{2} \setminus E(G))$. Two vertices u, v are called *adjacent* (in G) if $\{u, v\} \in E(G)$. We say that u and v are *incident* to $\{u, v\}$ and vice versa. We denote the *open neighborhood* of a vertex $v \in V(G)$ in G with $N_G(v) := \{u \in V(G) : \{u, v\} \in E(G)\}$ and the degree of v in G with $\deg_G(v) := |N_G(v)|$. The *closed neighborhood* $N_G[v]$ is $N_G(v) \cup \{v\}$. Accordingly, for a vertex set $S \subseteq V$ we set $N_G(S) := \bigcup_{v \in S} N_G(v)$ and $N_G[S] := N_G(S) \cup S$. We denote the *degree of a graph* G by $\Delta(G) := \max_{v \in G} \deg_G(v)$. The smallest d such that all subgraphs of G have a vertex with degree at most d is called the *degeneracy* of G.
A (simple) *path from u to w* in G is a sequence $P := (u = v_1, v_2, \ldots, v_\ell = w) \in V^\ell$ of vertices with $\{v_i, v_{i+1}\} \in E(G)$ for $i \in \{1, \ldots, \ell - 1\}$ and $v_i \neq v_j$ for $i \neq j$, where $\ell - 1$ is the *length* of the path. We define $V(P) := \{v_1, v_2, \ldots, v_\ell\}$. To emphasize that P starts in u and ends in w, we sometimes call P a *u-w-path*. A path with $u = w$ is called a *cycle*. Two vertices v and w are said to be *connected* in G if there is a v-w path in G. We use $\mathrm{dist}_G(v, w)$ to denote the length of a shortest path between v and w in G, also called *distance*. A subgraph G' of G whose vertices are pairwise connected is called a *connected component* of G. Each edge b of G such that $G - b$ has more connected components than G is called a *bridge* of G. A set $U \subseteq V(G)$ *separates* a vertex v from a set $U' \subseteq V(G)$ if the connected component of $G - U$ containing v does not contain any vertex in U'.
A central concept in this thesis is graph modification, that is, changing a graph into some other graph by deleting vertices, deleting or adding edges or other operations, like subdivision. Here, *subdividing* an edge $\{u, v\}$ means to delete the edge $\{u, v\}$ and introduce a new vertex w and new edges $\{u, w\}$ and $\{v, w\}$. The reverse of this operation is called *bypassing* w and is only defined if $\{u, v\} \notin E(G)$. For an edge $\{u, v\} \in E(G)$, the operation that deletes u and adds all edges in $\{\{v, w\} : \{u, w\} \in E(G)\}$ to the graph is called *contraction* of $\{u, v\}$. Note that, whenever some operation deletes a vertex u of a graph G, then it is implicit that all edges incident to u in G are also deleted. A graph that results from G by applying a series of edge contractions, edge deletions, and vertex deletions is called a *minor* of G.

If used without subscript, all sets and operators are defined with respect to a graph G that is clear from context.[5]

Directed Graphs. A directed graph (or digraph) is much like an undirected graph in that it contains vertices and connections between vertices. However, as indicated by the name, the connections are directed, that is, they are *ordered* pairs of vertices. To better distinguish between directed and undirected graphs, directed connections are called *arcs*, not edges, and are drawn as arrows instead of lines with the head of the arrow pointing to the second vertex of the pair. This vertex is called the *head* of the arc, while the first vertex is called the *tail* of the arc. In a directed graph $G = (V, A)$, a vertex v does not have neighbors but *predecessors* $\operatorname{pre}_G(v) := \{u : (u, v) \in A\}$ and *successors* $\operatorname{succ}_G(v) = \{w : (v, w) \in A\}$. Then, the *indegree* of v is $\operatorname{indeg}_G(v) := |\operatorname{pre}_G(v)|$ and the *outdegree* of v is $\operatorname{outdeg}_G(v) := |\operatorname{succ}_G(v)|$. If $\operatorname{indeg}_G(v) > \operatorname{outdeg}_G(v) = 0$, then we call v a *sink* of G. Likewise, if $\operatorname{outdeg}_G(v) > \operatorname{indeg}_G(v) = 0$, then we call v a *source* of G. The *complement* of $G = (V, A)$ is $\overline{G} := (V, V \times V \setminus A)$. In a digraph G, a vertex u is *strongly* connected to a vertex v if there is a path from u to v in G, that is, the transitive closure of A contains (u, v). The vertex u is *weakly* connected to v if u and v are connected in the *underlying undirected graph*, that is, the undirected graph obtained from ignoring the directions of the arcs of G. In analogy to undirected graphs, G has *weakly connected components* and *strongly connected components*. Most further concepts of undirected graphs are defined analogously on directed graphs.

Multigraphs. Allowing the edge set of a graph to be a multiset (that is, to contain an edge more than once) yields the concept of *multigraphs*, and, in analogy to the previous paragraph, *directed multigraphs*. Note that, in this setting, also the neighborhood or predecessors and successors, respectively, are multisets. Observe that there is no complement graph of a multigraph.

1.2.2 Graph Classes

Some properties of graphs are so important that they warrant special names for graphs fulfilling these properties (refer to Brandstädt et al. [32], de Ridder [162] for more details about graph classes). In this thesis, the following classes are most important.

[5] Also note that, for compatibility with Alber et al. [8], we write N^G instead of N_G in Chapter 3.

A *clique* is an undirected, complete graph, that is, each two vertices are adjacent. A clique with n vertices is denoted by K_n.

A *bipartite* graph is a graph whose vertices can be partitioned into two sets such that no edge is a subset of either set of the partition.

A *biclique* is a bipartite, complete graph, that is, each vertex is adjacent to all vertices that are not in the same set of the partition. A biclique whose partition has sets of sizes n_1 and n_2, respectively, is called a K_{n_1,n_2}.

A *tree* is a connected graph that does not contain cycles or, equivalently, a connected graph whose vertex set is strictly larger than its edge set. A degree-one vertex in a tree is called a *leaf*.

A *caterpillar* is a tree such that no vertex is adjacent to more than two non-leaves.

A *forest* is the disjoint union of trees.

A *planar* graph is a graph that can be "embedded" in a plane (that is, it can be drawn in a finite two-dimensional space with vertices as dots and edges as lines connecting these dots) such that no two edges cross or, equivalently, a graph that does not contain a K_5 or a $K_{3,3}$ as minor.

1.2.3 Computational Complexity

Instances and Problems. In order to examine the computational complexity of a problem, we first have to define what a computational problem is. To this end, consider a finite *alphabet* Σ (usually, $\Sigma = \{0, 1\}$) and define $\Sigma^0 := \{\lambda\}$, (where λ is said to be the *empty word*) and $\Sigma^m := \Sigma^{m-1} \times \Sigma$ for all $m \in \mathbb{N}^+$. Then, we call Σ^m the set of all words of length m. The set of all words is then $\Sigma^* := \bigcup_{m\in\mathbb{N}} \Sigma^m$. Now, an *instance* is an element of Σ^*. In large parts of this thesis, we will avoid this level of detail and just write "given a graph" when the input is a representation of a graph as a bitstring.

In this thesis, we mostly consider "decision problems" [10, 165]. A *decision problem* Q is a set of instances that fulfill some property. For example, the Even problem is the set of all bitstrings representing even non-negative integers. Equivalently, one can formulate the Even problem as "Given a number x, is x an even non-negative integer?". Instances of decision problems are often pairs of a graph G and some integer k. For example, the Vertex Cover problem is the set of all pairs (G, k), such that all edges of the graph G can be covered by at most k vertices or, equivalently, "Given a graph G and a number k, can we cover all edges of G by at most k vertices?". An instance I of a problem Q is called *yes-instance* if $I \in Q$ and *no-instance*, otherwise.

Vertex Deletion Problems. Many practical problems can be reformulated as vertex-deletion problems in graphs [99, 142, 202]. For example, the discussed

Vertex Cover problem can be stated as "given a graph G and an integer k, is it possible to make G edgeless by deleting at most k vertices?". Here, "being edgeless" is just one possible property of graphs. In general, with $\mathfrak{G}$ denoting a class of graphs, we can state the $\mathfrak{G}$-Vertex Deletion problem as follows.

$\mathfrak{G}$-Vertex Deletion
Input: An undirected graph G and an integer k.
Question: Can we delete at most k vertices of G such that the resulting graph is in $\mathfrak{G}$?

Many graph classes $\mathfrak{G}$ can be characterized using *forbidden subgraphs*, that is, a class $\mathfrak{F}$ such that for all graphs G, it holds that

$$G \in \mathfrak{G} \Leftrightarrow \forall_{\text{subgraph } H \text{ of } G} H \notin \mathfrak{F}.$$

For example, caterpillar forests can be characterized as graphs that do not contain a cycle or a 2-claw (three paths of length two that all share one common vertex, see Figure 2.2 on page 28). Analogously, some graph classes can be characterized using *forbidden induced subgraphs*. For example, the class of cluster graphs (disjoint unions of cliques) can be characterized as graphs that do not contain a path of length two as induced subgraph.

Running Times and Complexity Classes. We say that a decision problem Q can be *solved* in some time $T(n)$ if there is an algorithm that, given any instance x, determines whether $x \in Q$ using at most $T(|x|)$ computation steps.[6] It is common to refer to the length $|x|$ of the instance by n. Classical complexity theory defines a zoo of interesting classes of decision problems [4, 10, 165], only two of which are of interest to us here. The first class P of all problems that can be solved in "polynomial time", that is in n^c time for some $c \in \mathbb{N}$. The second class, called NP, is the class of all "polynomial-time verifiable" problems. Here, a problem Q is polynomial-time verifiable if there is a polynomial-time algorithm, that, given an instance x and a so-called "certificate", which is basically a proof for $x \in Q$, decides whether $x \in Q$ or not. More specifically, a problem Q is in NP if there is some function p (assigning certificates to instances) whose output size is polynomially bounded in its input size such that for each instance x of Q, the decision problem $\{(x, p(x)) : x \in Q\}$ is in P.

[6] Naturally, the number of steps needed depends on the employed machine model; in the scope of this thesis, we use the deterministic RAM (practically all modern computers are RAMs).

We assume the reader to be familiar with the Landau notation ("big-O notation") which is used to discard constant factors in functions and abbreviate

$$\text{poly}(n) := \bigcup_{i \in \mathbb{N}} \mathrm{O}(n^i).$$

Optimization vs. Decision. Note that there is the notion of "optimization problems" that actually produce optimal solutions instead of just determining whether a solution respecting a given condition exists. However, usually, an optimal solution can be constructed by slightly modifying the solution algorithm for the decision variant of a problem or by a limited (often linear in the input size) number of calls to the algorithm that solves the decision variant. For example, let G be a graph whose edges can be optimally covered by τ vertices and let consider an algorithm that decides $(G, k) \in$ Vertex Cover in $T(k, n)$ time. Then, by running the algorithm for increasing values of k, we can determine τ. Knowing τ, we can, for each vertex $v \in V(G)$, determine whether $G - v$ has a vertex cover of size $\tau - 1$ and if so, include v in a solution. The whole procedure then takes

$$T'(\tau, n) \leq \underbrace{\sum_{k=0}^{\tau} T(k, n)}_{\text{determine } \tau} + \underbrace{\sum_{k=0}^{\tau} n \cdot T(\tau - k, n)}_{\text{find vertices in the solution}} = (n+1) \cdot \sum_{k=0}^{\tau} T(k, n)$$

time. Assuming that $T(k, n)$ is exponential in k but not in n, we conclude $T'(\tau, n) \in \mathrm{O}(n\tau \cdot T(\tau, n))$. If $T(k, n)$ is even single exponential[7] in k, then geometric progression gives the promised linear relation $T'(\tau, n) \in \mathrm{O}(n \cdot T(\tau, n))$.

P vs. NP. The question of whether P = NP is both amazing and popular [94, 148] since many problems faced by people in many sciences and businesses are easily seen to be in NP but no algorithm solving them "fast", that is, in polynomial-time, is known.
Deciding whether or not P = NP is one of the seven famous "Millennium Prize Problems", stated by the Clay Mathematics Institute, a solution of each of which is promised to be rewarded with a US$1,000,000 prize [2]. The persistent inability to decide whether P = NP gave rise to the concept of polynomial-time reduction [10, 165]. We say that a problem A can be (polynomial-time) reduced to a problem B if there is some algorithm running in polynomial time and, given an instance x of A produces an instance y of B such that $x \in A$ if and only if $y \in B$. This allowed the concept of so-called "hard problems" for the class NP: A problem Q is said to be NP-*hard* if all problems in NP can be reduced to it. Over

[7] The function $T(k, n)$ is *single exponential* in k if $T(k, n) = c^k \cdot \text{poly}(n)$ for some $c \in \mathbb{N}$.

the last decades, thousands of problems have been shown to be NP-hard [100] (mostly by using the transitivity of reducibility that allows showing NP-hardness of a problem Q by reducing any NP-hard problem to Q). Since it is widely believed [101], we will work under the hypothesis P $\neq$ NP in this thesis.

Coping with NP-hardness. Many real-world problems have been shown to be NP-hard and, thus, do not admit polynomial-time algorithms. Since we can use algorithms that solve optimization problems to solve decision problems, this also implies that there are no polynomial-time algorithms to minimization or maximization problems corresponding to NP-hard decision problems. Since we cannot simply refuse to solve these problems, ways of coping with this computational hardness have developed. One possibility is to accept a "fuzzy" solution, that is, a feasible solution that is not necessarily optimal, but can be shown to be reasonably close to an optimal solution. Many problems have been shown to admit such "approximation" algorithms of various degrees of fuzziness [11, 199, 192]. A more recent approach to coping with NP-hard problems exploits the fact that the instances produced by reductions are pathological for most applications, that is, aspects that make the considered problem hard, are often unlikely to be seen in practice. Thus, it might be possible to design an algorithm that is fast for instances that are practically relevant, but takes exponential time for other instances. A theoretic approach incorporating this thought is "parameterized complexity" [66, 89, 160]. Another theoretic approach in this direction is the (relatively new) technique of "smoothed analysis" [179, 180], that diverges from the concept of worst-case analysis, which "is often the source of discrepancy between the theoretical evaluation of an algorithm and its practical performance" [180]. Here, the performance of algorithms is measured as expected running time when input instances are exposed to a slight random perturbation, thus eliminating the influence of pathological worst-case instances.

1.2.4 Parameterized Complexity

Our results are in the context of parameterized complexity, which is a two-dimensional framework for studying computational complexity [66, 89, 160]. One dimension is the input size, and the other one is the *parameter* (usually a positive integer). In this context, a *parameterized problem* is a subset of $\Sigma^* \times \mathbb{N}$. If not explicitly stated, we assume that the parameter for a parameterized problem Q is called k. A problem is called *fixed-parameter tractable* (fpt) with respect to a parameter k if it can be solved in $O^*(g(k))$ time, where g is a computable function only depending on k. An algorithm for the problem that achieves this running time is called *fixed-parameter algorithm*.

Definition 1.1. *Let $g, g' : \mathbb{N} \to \mathbb{N}$ be computable functions. A parameterized problem Q_1 is* parameterized reducible *to a parameterized problem Q_2 if there is an algorithm that, given an instance (x, k) of Q_1, produces an instance (x', k') of Q_2 in $O^*(g(k))$ time such that*

1. *$(x, k) \in Q_1 \Leftrightarrow (x', k') \in Q_2$ and*
2. *$k' \leq g'(k)$.*

If Q_1 is parameterized reducible to Q_2 and vice versa, then Q_1 and Q_2 are parameterized equivalent.

To improve readability, we abbreviate "Q parameterized by k" to "k-Q" where appropriate.[8] For example, Vertex Cover parameterized by the treewidth of the input is denoted by tw-Vertex Cover. Furthermore, to avoid having to drag "unparameterized versions" of parameterized problems along, we say that a parameterized problem k-Q is in NP or NP-hard if the problem $\{x\#1^k : (x, k) \in k\text{-}Q\}$, where 1^k denotes a unary encoding of k [26] is in NP or NP-hard, respectively.
Much like the NP-completeness theory in classical complexity, there is a class of problems that is conjectured to not allow for fixed-parameter algorithms. In fact, there is a fully-grown hierarchy of classes of problems. For our purposes, it is sufficient to define the "first level" in this hierarchy, which is called $W[1]$. Like in classical complexity, this class can be defined with a problem which we call "complete" for $W[1]$ and the concept of parameterized reduction defined above. A canonical candidate for the complete problem for $W[1]$ is the following.

k-Clique
Input: An undirected graph G and an integer k.
Question: Does G contain a clique on k vertices as subgraph?
Parameter: k

We call a parameterized problem Q $W[1]$-*hard* if k-Clique is parameterized reducible to Q. Parameterized problems that are $W[1]$-hard are conjectured to not admit fixed-parameter algorithms [66, 89, 160]. Thus, the range of parameter values for which the problem can be solved in polynomial time is much smaller. There are several strategies of developing fixed-parameter algorithms, some of which we use in this thesis and should therefore be discussed here. Note that this is only a small extract of the versatile toolbox of parameterized algorithmics.

Bounded Search-Trees. Recall the notion of vertex deletion problems, where a given graph is to be transformed to a graph of a given graph class $\mathfrak{G}$ by

[8] In this context, "k" is to be considered as a string containing the name of the variable that serves as parameter, not the variable itself. Hence, the problem k-Q is still called k-Q even if an instance with $k = 2$ is considered.

deleting some of its vertices. If, for some $\mathfrak{G}$, the forbidden subgraph characterization is finite, then a strategy for deciding whether an instance (G, k) is in $\mathfrak{G}$-Vertex Deletion is the following:

Step 1: Find a forbidden subgraph H in the input graph G. If no such subgraph exists, then return that $(G, k) \in \mathfrak{G}$-Vertex Deletion.

Step 2: If $k = 0$, then return that $(G, k) \notin \mathfrak{G}$-Vertex Deletion.

Step 3: For each vertex v of H, return the result of recursively asking whether $(G - v, k - 1)$ is in $\mathfrak{G}$-Vertex Deletion.

Each recursive call of this algorithm can be interpreted as a vertex in a tree-structure that completely describes the process of running this algorithm. We call this structure *search-tree* and, correspondingly, the strategy is called *branching strategy*.
Notably, if we can find such subgraphs in polynomial time, this yields a fixed-parameter algorithm for the parameterized version k-$\mathfrak{G}$-Vertex Deletion running in $O^*((\max\{|V(H)| : H \in \mathfrak{F})^k)$ time [37]. It is often possible to achieve better running times by exploiting observations concerning the set of forbidden subgraphs. In this case, a number of "branching rules" can be used.

Definition 1.2. *Let $\ell \in \mathbb{N}$ and let Q be a parameterized problem. Then, a* branching rule *R is a polynomial-time algorithm that, given an input instance (x, k) of Q computes a set of ℓ instances (x_i, k_i) with $0 \leq i < \ell$ of Q. We say that R is* correct *if*

1. $(x, k) \in Q \Leftrightarrow \exists_{0 \leq i < \ell}(x_i, k_i) \in Q$ *and*
2. $\forall_{0 \leq i < \ell}\, k_i < k$.

Fixpoints of R (that is, instances on which R performs no operations) are called reduced with respect to *R. The procedure of repeatedly applying a branching rule R is called* exhaustive application *of R. The vector $(k - k_0, k - k_1, \ldots, k - k_{\ell-1}) \in \mathbb{N}^+$ is called the* branching vector *of R. We consider all branching vectors with ℓ components ordered by the canonical componentwise "$\leq$" operator.*

For ease of presentation, we allow branching rules to output "partial solutions" instead of new instances if it is clear how an instance results from applying a partial solution to the input instance. Note that the condition that $k_i < k$ for all $0 \leq i < \ell$ is needed to ensure that the search-tree is finite. However, it is reasonable to drop this condition if we can ensure that the number of times this branch is chosen on a computation path to a leaf of the search tree can be bounded by a function in k.
For example, a simple forbidden subgraph characterization for edgeless graphs is $\{K_{1,1}\}$ giving rise to the following branching rule for Vertex Cover.

Branching Rule 1.1. *Let $\{u, v\}$ be a $K_{1,1}$ subgraph of the input graph G. Then, create the partial solutions $\{u\}$ and $\{v\}$.*

Clearly, Branching Rule 1.1 has a branching vector of $(1, 1)$. It is easy to see that this corresponds to a search-tree of size $O(2^k)$.

Dynamic Programming. When considering branching algorithms in general, it is often observed that many recursive questions near the leaves of the search-tree are repeated. For example, if a computation path in the search-tree first tries to delete some vertex v and in the next step tries deleting another vertex u, the following recursion produces the same result as if u was deleted first and then v in the next step since $(G-v)-u) = (G-u)-v$. It is reasonable to try and safe time by storing the result of certain recursive calls, essentially "pruning the search tree" [91]. Dynamic programming often comes in the form of computing the entries of a table with the help of other table entries. Finally the solution to the input instance can be read from the final table-entry. The presentation of a dynamic programming algorithm usually consists of stating the semantics of a table entry and then stating the formula for computing an entry from previously computed entries. This way, proving that the formula matches the stated semantics is usually sufficient to prove correctness of the algorithm. To emphasize the versatility of dynamic programming, we give two discriminative examples.

- The Coin Change problem is, given a set D of n types of coin denominations, an amount a of money and an integer k, determine whether making change for the amount a is possible with at most k coins. A standard way of solving this problem is to compute a one-dimensional table T such that $T[x]$ contains the minimum number of coins needed to make change for x (semantics of T). To compute $T[x]$ for increasing values of x, first initialize $T[d] = 1$ for each $d \in D$. Then, compute $T[x] := 1 + \min\{T[x - d] : d \in D \wedge d \leq x\}$ for all $x \leq a$ in increasing order. Then, the initial question is equal to "$T[a] \leq k$?" and the algorithm runs in $O(a \cdot n)$ time.
- A breakthrough result for the NP-hard Traveling Salesman problem was given by Held and Karp [115]. Traveling Salesman can be stated as follows.

  Traveling Salesman
  Input: A clique C on n vertices, an edge-weight function $\omega : \binom{V(C)}{2} \to \mathbb{N}$, and an integer k.
  Question: Is there a cycle of length n in C such that the total sum of edge weights used in the cycle is at most k?

Prior to the dynamic programming algorithm of Held and Karp [115], only the trivial $O(n!)$-time algorithm was known.

In terms of semantics, a table entry $T[S, u]$ contains the smallest weight of a path from a dedicated vertex v to u containing exactly the vertices in S. To compute the entries, first initialize $T[\{u, v\}, u] := \omega(\{u, v\})$ for each $u \in V(C)$ and then use the formula $T[S, u] := \min\{T[S \setminus \{u\}, x] + \omega(\{x, u\}) : x \in S \setminus \{u\}\}$ for increasing sizes of S and each $u \in S$. The answer to the initial question is then equivalent to "$T[V(C), v] \leq k$?" and the algorithm runs in $O(2^n \cdot n^2)$ time.

Kernelization. A core tool in the development of fixed-parameter algorithms and a central topic in this thesis is polynomial-time preprocessing by *data reduction* [23, 105, 145]. Here, the goal is to transform a given instance (x, k) of a parameterized problem Q into an equivalent instance (x', k') of Q such that the size of x', measured in k, is small. The process of simplifying an input like this is called *kernelization*.

Definition 1.3. *Let Q be a parameterized problem and let $g, g' : \mathbb{N} \to \mathbb{N}$ be computable functions. An algorithm that, given an instance (x, k) of Q, computes an instance (x', k') of Q in polynomial time such that*

1. $(x, k) \in Q \Leftrightarrow (x', k') \in Q$,
2. $|x'| \leq g(k)$, *and*
3. $k' \leq g'(k)$

is called a kernelization *for Q. The result produced by the algorithm is called a* (problem) kernel *for Q. We call $g(k)$ the* size *(in bits) of the problem kernel.*

There are multiple definitions of kernelization in the literature. Maybe the largest dispute is about whether or not to require $k' \leq k$ instead of $k \leq g'(k)$, as, in the spirit of parameterized complexity, a blowup of the parameter actually makes the instance *harder* to solve. On the other hand, it is reasonable to accept a moderate blowup of the parameter in order to remove large parts of the input. Chen et al. [48] call a kernelization with $k' \leq k$ *strong* and show that the two definitions do not coincide unless P = NP. In the scope of this thesis, however, the difference does not matter. When writing "Q admits a problem kernel of size $O(k^2)$", we mean that a kernelization with $g(k) \in O(k^2)$ exists for Q.
Although Definition 1.3 formally allows the size of a problem kernel to be exponential in the parameter, polynomial-size problem kernels are considered way more interesting and practical. In particular, every (decidable) fixed-parameter tractable problem admits a problem kernel and vice versa [39]. This is due to the fact that, if $f(k) < n$, then $O^*(f(k))$ is polynomial in n and, hence, either $n \leq f(k)$

or we can solve the instance in polynomial time (that is, construct a constant-size instance that is equivalent to the input instance), yielding a problem kernel of size $f(k)$. Thus, it is much more interesting to develop a *polynomial-size* problem kernel than just any problem kernel.
It is customary to present a kernelization by giving a set of polynomial-time applicable "(data) reduction rules". Reduction rules are basically branching rules (see Definition 1.2) that create just one instance. However, since the concept is of central importance, we will briefly state the modified version of Definition 1.2.

Definition 1.4. *A* (data) reduction rule *R is a polynomial-time algorithm that, given an input instance (x, k) of a parameterized problem Q computes an instance (x', k') of Q. We say that R is* correct *if*
1. $(x, k) \in Q \Leftrightarrow (x', k') \in Q$ *and*
2. $k' \leq k$.

Fixpoints of R are called reduced with respect to *R.*

Note that the requirements 1 and 3 of Definition 1.3 are ensured by correct reduction rules. For example, the following data reduction rules by Buss [35] are enough to provide a problem kernel of size $O(k^2)$ for Vertex Cover with parameter k:

Reduction Rule 1.1. *Let v be a vertex.*
1. *If* $\deg(v) > k$*, then delete v and decrease k by one.*
2. *If* $\deg(v) = 0$*, then delete v.*

Reduction Rule 1.2. *Let* $\deg(v) \leq k$ *for all vertices v. If the input contains more than k^2 edges, then return a constant-size no-instance of* Vertex Cover*, e.g.,* $(K_2, 0)$.

Since for all vertices v, each vertex cover contains v or $N(v)$, a vertex cover of size k contains all vertices v with $|N(v)| > k$. Thus, Reduction Rule 1.1 is correct. Since for all vertex covers C, all edges are incident to vertices in C, a size-k vertex cover in a graph with maximum degree k cannot cover more than k^2 edges. Thus, Reduction Rule 1.2 is correct. To see that these rules yield a problem kernel for k-Vertex Cover, consider an instance $((G, k), k)$ that is reduced with respect to them. Clearly, by reducedness with respect to Reduction Rule 1.1 we conclude that the degree of each vertex v in G is $1 \leq \deg(v) \leq k$. Hence, G contains at most k^2 edges since, otherwise, Reduction Rule 1.2 would apply to G, contradicting its reducedness.
Since the notions of branching and data reduction rules are so close, kernelization integrates well with branching algorithms. Consider for example Branching Rule 1.1. Using the fact that solving Vertex Cover on graphs of maximum degree two is trivial, we can derive the following data reduction rule.

Reduction Rule 1.3. *Let* (G, k) *denote an input instance such that* $\Delta(G) = 2$. *If* $(G, k) \in$ Vertex Cover, *then return a constant-size yes-instance of* Vertex Cover, *e.g.,* $(K_2, 1)$. *Otherwise, return a constant-size no-instance of* Vertex Cover, *e.g.,* $(K_2, 0)$.

This allows us to restrict application of Branching Rule 1.1 to vertices with at least three neighbors.

Branching Rule 1.2. *Let* v *be a vertex in the input graph such that* $\deg(v) \geq 3$. *Then, create the partial solutions* $\{v\}$ *and* $N(v)$.

Clearly, Branching Rule 1.2 has a branching vector of at least $(1, 3)$. It can be shown [87, 91] that this corresponds to a search-tree of size $O(1.47^k)$. Since all instances of Vertex Cover that are reduced with respect to both Reduction Rule 1.3 and Branching Rule 1.2 have constant size, we conclude that these two rules are enough to solve Vertex Cover in $O^*(1.47^k)$ time.

Kernelization Lower Bounds. The question whether all fixed-parameter tractable problems have polynomial-size problem kernels was only recently answered negatively [26, 95] (under the assumption that the polynomial hierarchy does not collapse). The central technique behind this result is called *OR-composition*. Loosely speaking, a polynomial-size problem kernel for an NP-hard problem k-Q is unlikely if multiple instances x_i of k-Q can be assembled into one instance of k-Q that is equivalent to the OR of the x_i instances and whose parameter value is small. This is reasonable, since if such an OR-composition exists and k-Q has a small (polynomial-size) kernel, then the OR of any number of instances of an NP-hard problem could be represented in small space. A more easily applicable version of this technique, that also allows representing the OR of instances of a different NP-hard problem, was given by Bodlaender et al. [29].

Definition 1.5 ([29])**.** *An equivalence relation* R *on instances of a problem* L *is called* polynomial *if*

1. *equivalence of two instances can be checked in time polynomial in their sizes and*
2. *the elements of any finite set* S *of instances of* L *are partitioned into at most* $\text{poly}(\max\{|x| : x \in S\})$ *equivalence classes.*

Definition 1.6 ([29])**.** *Let* L *be an* NP*-hard problem, let* Q *be a parameterized problem, and let* R *be a polynomial equivalence relation on instances of* L. *A polynomial-time algorithm that, given* t *instances* x_i, $0 \leq i < t$, *of* L *that are equivalent under* R *computes an instance* (x, k) *of* Q *such that*

1. $(x, k) \in Q \Leftrightarrow \exists_{0 \leq i < t}\, x_i \in L$ *and*

2. $k \in \text{poly}\left(\max\{|x_i| : 0 \leq i < t\} + \log t\right)$

is called cross composition *of L into Q. If such a composition exists for some Q and L, then Q is called* compositional.

Informally speaking, a cross composition gets a number of instances of an NP-hard problem that can be assumed to be equal under some polynomial equivalence relation. After polynomial processing time, it outputs an instance of Q that is a yes-instance if any of the input instances is a yes-instance (and vice versa).
For example, consider the Clique problem with respect to the parameter "maximum degree Δ of the input graph", which is fixed-parameter tractable [117]. A simple cross composition of Clique into Δ-Clique is the following: First, define a polynomial equivalence relation R that relates two instance if and only if they have the same parameter value and ask for the same clique size k. Then, just output the disjoint union of all input instances. Clearly, for all k, a k-clique can be found in one of the instances if and only if it can be found in the disjoint union. Since the maximum degree of the output instance is not larger than the size of any input instance, the condition 1.6(2) is satisfied (we do not even require the additional $\log t$ summand in this example).
Bodlaender et al. [29] showed that the existence of a cross composition into a parameterized problem Q excludes polynomial-size problem kernels for Q, unless an unexpected complexity-theoretic collapse occurs.

Theorem 1.1 ([29])**.** *Let Q be a parameterized problem and let L be an* NP*-hard problem such that there is a cross-composition of L into Q and Q admits a polynomial-size problem kernel. Then,* $\text{NP} \subseteq \text{coNP/ poly}$.

In this thesis, we accept $\text{NP} \subseteq \text{coNP/ poly}$ as an unlikely event without further explanation of its complexity-theoretic implications (A brief discussion can be found in Appendix A).
In the recent past, many parameterized problems have been shown to not admit a polynomial-size kernel unless $\text{NP} \subseteq \text{coNP/ poly}$. As mentioned by Dom et al. [61], lower bounds shown by this framework also imply the inability to "compress" instances to polynomial size [113]. Herein, a compression algorithm is very similar to a polynomial-size kernelization with respect to the parameter $c + \log |x|$, where c is the size of a smallest certificate for the input instance x (see Section 1.2.3).
This year, Drucker [68] showed that Theorem 1.1 also holds for AND-compositions, which are defined analogously to OR-compositions, with the difference that the output instance is a yes-instance if and only if *all* input instances are yes-instances. Especially for the parameter treewidth, polynomial-size problem kernels can be excluded using AND-composition. As an example, consider the Vertex Cover

problem with parameter "treewidth of the input graph" (tw-Vertex Cover). It is known [21] that the infamous NP-complete 3SAT problem[9] can be reduced to Vertex Cover, producing instances (G, k) such that G does not have a vertex cover of size $k - 1$. Let us denote Vertex Cover restricted to instances produced by said reduction as Satisfying Vertex Cover. A simple AND-composition for tw-Satisfying Vertex Cover is the following: Given t instances (G_i, k_i) for $1 \leq i \leq t$, compute the disjoint union of all input graphs G_i and ask for a vertex cover of size $\sum_{1 \leq i \leq t} k_i$. Thus, tw-Satisfying Vertex Cover and, since it is a generalization, also tw-Vertex Cover, does not admit a polynomial-size kernel unless NP $\subseteq$ coNP/ poly. Likewise, other problems on graphs including the Treewidth problem itself can be shown to be unlikely to admit a polynomial-size problem kernel with respect to the treewidth of the input graph [30, 145]. This especially motivates considering kernelization with respect to structural parameters that are "weaker" than (or, equivalently, lower-bounded by) the treewidth of the input graph.

Note that hardness of (polynomial-size) kernelization can also be derived by reduction. Here, the reduction concept is a special kind of parameterized reduction (see Definition 1.1) that additionally requires the reduction to run in polynomial time and the parameter of the output instance to be polynomially bounded in the parameter of the input instance (in terms of Definition 1.1, $g'(k) \in \text{poly}(k)$). This reduction is called *polynomial-time polynomial-parameter reduction* [28]. Recall our example for Δ-Clique and note that it also implies that Clique does not admit a polynomial-size kernel with respect to the parameter "chromatic number χ of the input"[10] since $\chi \leq \Delta + 1$ for all graphs. Then, the following is a polynomial-time polynomial-parameter reduction of χ-Clique to Vertex Cover with parameter "vertex-clique cover number ψ of the input graph"[11]: Given an instance $((G, k), \chi)$ of χ-Clique, output $((\overline{G}, |V(G)| - k), \psi)$, where $\overline{G}$ is the complement of the input graph G and, since color classes of colorings of G and cliques in $\overline{G}$ are equivalent, $\psi = \chi$. Since a k-clique in G is a k-independent set in $\overline{G}$, its complement is a vertex cover of size $|V(G)| - k$ for $\overline{G}$. Hence, the construction is a polynomial-parameter reduction and, hence, implies that ψ-Vertex Cover does not admit a polynomial-size problem kernel.

[9] Given a boolean formula in conjunctive normal form with at most three literals per clause, decide whether a satisfying assignment of variables to true or false.

[10] The chromatic number of a graph is the minimum number of colors needed to color each vertex such that no two adjacent vertices have the same color.

[11] The vertex clique cover number of the input graph is the smallest number of cliques in the input such that each vertex in the input is in one of the cliques.

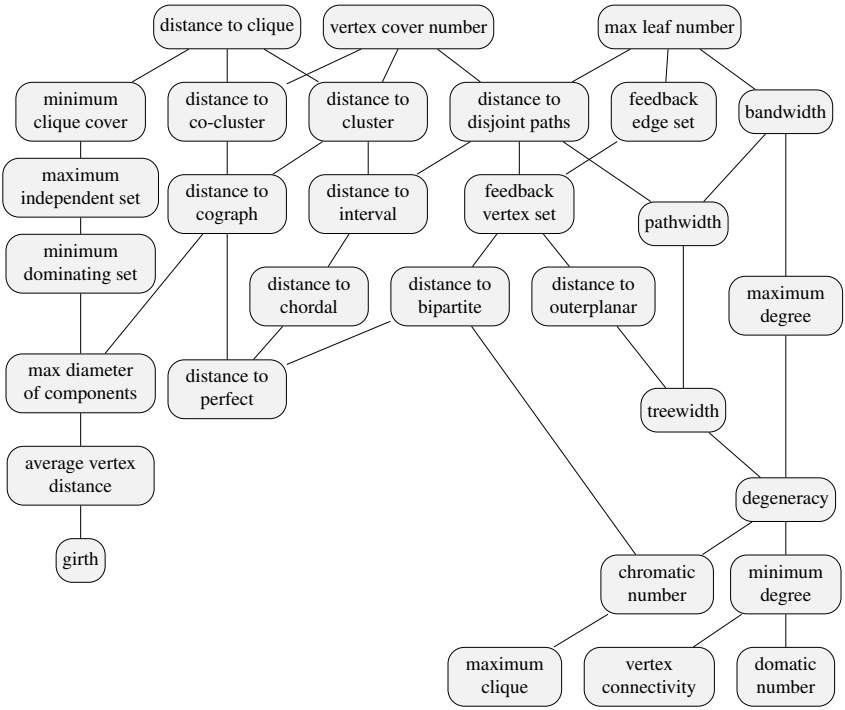


Figure 1.1: Diagram showing the $\leq_{\text{poly}}$-relations between important graph parameters. An edge between two parameters means that the upper parameter can be lower-bounded by a polynomial in the lower parameter. For example, the maximum degree Δ of a graph G can be lower bounded by a (linear) function in the degeneracy d of G, that is, $d \leq_{\text{poly}} \Delta$. In this sense, the lower the position of a parameter, the "stronger" the parameter. Refer to Diestel [60] for definitions of these parameters. Here, "distance to" refers to the vertex-deletion distance, that is, the minimum number of vertices that has to be removed to obtain a graph of the specific class.

Parameter Hierarchy. In order to choose a good parameter for a problem, it is important to be aware of relations between parameters. If, for example, the problem we are considering has been show to be $W[1]$-hard with respect to the pathwidth $\mathrm{pw}(G)$ of the input graph G, then it is also $W[1]$-hard with respect to the treewidth $\mathrm{tw}(G)$ of G, as $\mathrm{pw}(G) \geq \mathrm{tw}(G)$ for all graphs G. To draw such a conclusion, it is enough to upper bound $\mathrm{tw}(G)$ by *any function* in $\mathrm{pw}(G)$. Likewise, if a problem is known to be fixed-parameter tractable with respect to the pathwidth, then it might be interesting to consider the parameter "treewidth of the input graph", or even "degeneracy of the input graph".
In this thesis, we want to focus on preprocessing in general, and polynomial-size kernelizations in particular and, hence, we are particularly interested in polynomial bounds. Thus, considering parameters as functions mapping inputs to integers, we define the relation $\leq_{\mathrm{poly}}$ such that

$$\forall_{f_1,f_2:\Sigma^*\to\mathbb{N}} f_1 \leq_{\mathrm{poly}} f_2 \Leftrightarrow f_1 \in \mathrm{poly}(f_2).$$

Throughout the thesis, we will usually write "f_1 is a *stronger parameter* than f_2" instead of $f_1 \leq_{\mathrm{poly}} f_2$. The relation $\leq_{\mathrm{poly}}$ gives rise to a natural hierarchy of parameters. This hierarchy, restricted to some common graph parameters, is partially depicted in Figure 1.1. The diagram is an extension of the diagram presented by Jansen [122]. Note that the size of a maximum clique is among the strongest parameters shown.

Chapter 2

Non-standard Parameters

In this chapter, we present an example of a kernelization for a non-standard parameter. In fact, the parameter can be considered "stronger" than the standard parameter, in the sense that it lower-bounds the standard parameter, but can be arbitrarily smaller in practical instances. More precisely, we consider the Two-Layer Planarization problem with respect to the parameter "size f of a minimum feedback edge set of the input". Roughly speaking, Two-Layer Planarization asks for a smallest set of edges to remove from a given graph such that the remainder is a forest of special trees (caterpillars). We present a kernelization yielding a problem kernel of size $O(f)$. To date, we know of little other problems for which polynomial problem kernels with respect to structural parameters are known:

1. Treewidth with respect to the parameter "size of a smallest vertex cover of the input graph" and "size of a smallest feedback vertex set of the input graph" [30],
2. Graph Coloring with respect to the parameter "size of a smallest vertex cover of the input graph" and "vertex-deletion distance to cographs[12] or C-split graphs [13]" [123].
3. Odd Cycle Transversal with respect to the parameter "vertex-deletion distance to bipartite graphs of constant treewidth" [124].
4. Vertex Cover with respect to the parameter "size of a smallest feedback vertex set of the input graph" [125], and
5. Target Set Selection with respect to the parameter "size of a minimum feedback edge set of the input graph" [159].

We supplement the kernelization with two branching algorithms that solve Two-Layer Planarization. The asymptotic running time of one of them is comparable to the running time of the state-of-the-art algorithm with respect to the standard parameter. Our choice of parameter is motivated as follows.

[12] A cograph is a graph that can be colored with χ colors if and only if there is no $K_{\chi+1}$.

[13] A C-split graph is a graph whose every connected component is a split graph, that is, each component can be partitioned into an independent set and a clique.

f is stronger than the standard parameter. The size k of a solution of an instance of Two-Layer Planarization is at least the feedback edge set number of the input. In this sense, we improve on previous results [69, 182] by providing fixed-parameter algorithms and kernelizations with about the same worst-case bounds for a parameter that we expect to be significantly smaller for a wide range of input instances. For large values of k and small values of f, our 3.8^f poly(n)-time search tree algorithm may in fact outperform the state of the art 3.652^k poly(n)-time algorithm by Suderman [182].

f is a good measure for sparseness. Dujmović et al. [69] pointed out that "instances of Two-Layer Planarization for *dense* graphs are of little interest from a practical point of view" since the resulting drawings are unreadable anyway. They argue that this indicates small solution sizes in practice. However, even for trees, the sparsest connected graphs, the solution size can be arbitrarily large. Thus, a parameter that is directly linked to the sparseness, as the treewidth, the feedback vertex set number or the feedback edge set number, supports exploiting the above observation even better. Measuring the distance from trees by the feedback edge set number can be seen as a parameterization by "distance from triviality" [106]. In this sense, our results generalize the linear-time algorithm for trees by Shahrokhi et al. [174].

f can be precomputed. The feedback edge set number f is a parameter that can easily be computed in advance and, hence, allows for a meta-algorithm that chooses an algorithm for a given input by computing an estimation on the running time prior to running the algorithm for the problem itself. Since the parameter k ("number of edge deletions") is NP-hard to compute, such an algorithm could not efficiently determine the running time of an algorithm parameterized by k in advance.

f is an upper bound for the treewidth. One of the most prominent structural parameters in parameterized algorithmics for graph problems is the treewidth of the input graph. However, for a large set of problems, the treewidth of the input might be too strong a parameter to allow polynomial-size kernels (see Section 1.2.4). Thus, parameters that are lower bounded by the treewidth, like the vertex cover number, the feedback vertex set number, and, last but not least, the feedback edge set number, have recently been considered [83, 84, 125, 137, 159].

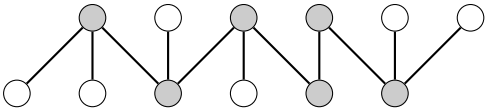
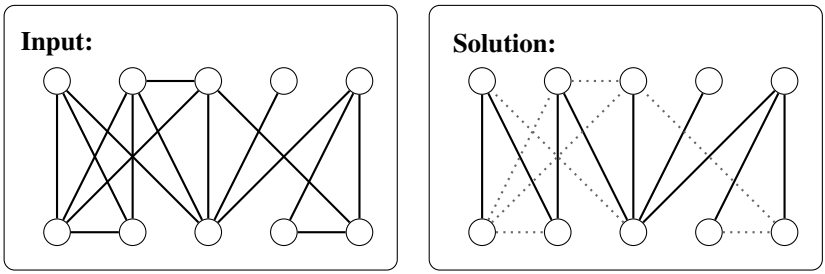


Figure 2.1: A connected graph that is biplanar and, hence, a caterpillar tree. White vertices are leaves, gray vertices are non-leaves. Note that no vertex has more than two non-leaf neighbors.

2.1 Introduction to Two-Layer Planarization

To formally define Two-Layer Planarization, consider an arrangement of the vertices of a graph G on two parallel lines called layers. Then, this drawing is called *biplanar* if no edge runs between two vertices of the same layer and, when drawn as straight lines between the layers, no two edges of G cross. Note that, equivalently, the edges can be drawn in an arbitrary fashion (not just as straight lines) as long as they never cross one of the two parallel lines). We say that the graph G is biplanar if it admits a biplanar drawing. Then, Two-Layer Planarization can be defined as follows.

Two-Layer Planarization
Input: An undirected graph $G = (V, E)$ and an integer $k \geq 0$.
Question: Is there an edge set $E' \subseteq E$ such that $|E'| \leq k$ and $G \dot{-} E'$ is biplanar?



It has been shown that biplanar graphs are exactly the graphs that consist of disjoint caterpillar trees (or caterpillars for short) [112]. A caterpillar is a tree such that each of its vertices is adjacent to at most two non-leaf vertices, see Figure 2.1. This allows us to equivalently define Two-Layer Planarization as follows.

Two-Layer Planarization
Input: An undirected graph $G = (V, E)$ and an integer $k \geq 0$.
Question: Is there an edge set $E' \subseteq E$ such that $|E'| \leq k$ and $G \dot{-} E'$ is a forest of caterpillars?

Two-Layer Planarization is motivated mainly by its application in graph drawing, where it is an integral part of the "Sugiyama approach" [184, 156] to multilayered graph drawing [69, 86, 133]. Here, the basic strategy is to assign all vertices to different layers, solve Two-Layer Planarization to get an optimal layout of the first two layers and repeatedly solve 1-Layer Planarization to layout subsequent layers. In 1-Layer Planarization, the arrangement of vertices on one of the layers is already given and the task is to remove as little edges as possible such that the remaining vertices can be arranged on the second layer without edge crossings. Apart from graph drawing, solving Two-Layer Planarization is important in DNA mapping [195] and global routing for row-based VLSI layout [139].

2.1.1 Previous and Related Work

Two-Layer Planarization is NP-hard even in the case that the input graph is bipartite and in one partition each vertex has degree at most two [71]. Note that Eades and Whitesides [71] called the problem Maximum Biplanar Subgraph. Shahrokhi et al. [174, Section 6] presented a dynamic programming based linear-time algorithm solving the problem on trees. Concerning the parameter k (number of edge deletions), Dujmović et al. [69] showed that Two-Layer Planarization can be solved in $O(6^k \cdot k + |G|)$ time by devising a search tree algorithm and several polynomial-time data reduction rules leading to a problem kernel comprising $O(k)$ vertices and edges. Later, Fernau [86] presented a refined search tree for Two-Layer Planarization leading to a running time of $O(5.19276^k \cdot k^2 + |G|)$. Finally, based on a different branching analysis, Suderman [182] developed an $O(3.562^k \cdot k + |G|)$-time algorithm, which he tested extensively on different types of sparse graphs.
Dujmovic et al. [70] considered a multilayered problem version, where the task is to transform the input graph into a graph that can be drawn in h layers without edge crossings. They developed a path decomposition based algorithm that runs in $g(h, k) \cdot |V|$ time (where the function g is not explicitly specified).
Two-Layer Planarization is a special case of the problem of transforming a binary matrix into a matrix with so-called "consecutive-ones property"[14] by a

[14] A binary matrix is said to have the consecutive-ones property if its columns can be permuted such that, in each row, all ones occur consecutive.

minimum number of column removals. More specifically, Two-Layer Planarization coincides with this problem for matrices without identical columns that have a maximum of two 1s in each column [185, 63]. Dom et al. [63] showed approximability and fixed-parameter tractability results for these related submatrix problems.
Finally, observe that caterpillars are exactly the graphs with pathwidth at most one [167]. Thus, Two-Layer Planarization is the problem of transforming a given graph into a graph with pathwidth at most one by deleting edges. Under the name Pathwidth One Vertex Deletion, Cygan et al. [56] considered the vertex-deletion variant of Two-Layer Planarization, developing a quadratic-size problem kernel for the parameter "number of allowed vertex deletions".

2.1.2 New Results

We investigate the parameterized complexity of Two-Layer Planarization with respect to the structural parameter "feedback edge set number" f, that is, the minimum number of edges whose removal results in an acyclic graph. Note that the feedback edge set number of a connected n-vertex and m-edge graph G is $f(G) = m - n + 1$ and a minimum feedback edge set can be determined by the computation of a spanning tree in $O(n + m)$ time via depth-first search. We develop efficient preprocessing rules for Two-Layer Planarization that lead to a problem kernel with $O(f)$ vertices and edges.
We present a new search tree algorithm leading to a total running time of $O(6^f \cdot f^2 + (f + 1) \cdot |G|)$ for solving Two-Layer Planarization. Improving on this running time, we develop a search-tree algorithm running in $O(3.8^f f^2 + f \cdot |G|)$ time. We implemented and tested this algorithm, augmented by several heuristic improvements, for the instances used by Suderman [182] to test his algorithm and some sparser graphs.

2.2 Preliminaries

For a graph G, let $I(G)$ (isolated vertices) and $L(G)$ (leaves) denote the set of vertices in G with degree zero and one, respectively. For a vertex v in G, let $L(v) := L(G) \cap N(v)$. We define the *non-leaf degree* $\widehat{\deg}_G(v) := |N_G(v) \setminus L(G)|$ for every vertex $v \in V(G)$ (see [69]).
A tree is a *caterpillar tree* (or caterpillar for short) if each of its vertices has non-leaf degree at most two. Equivalently, a caterpillar is a tree that does not contain a 2-claw [71] (see Figure 2.2). Thus, unions of caterpillars have the following forbidden-subgraph characterization: A graph is a forest of caterpillars if and only

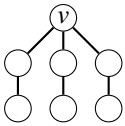


Figure 2.2: A 2-claw with center vertex v. Caterpillars can be characterized as graphs containing neither cycles nor 2-claws.

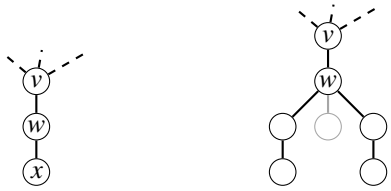
if it is acyclic and does not contain a 2-claw as subgraph. Note that for a caterpillar tree, the non-leaf degree of a vertex is at most 2. A leaf $v \in L(G)$ is called *critical* if its only neighboring vertex has non-leaf degree two. The definition of critical vertices is motivated by the observation that being a caterpillar is invariant with respect to adding neighbors to non-critical vertices. In contrast, adding a neighbor to a critical vertex creates a 2-claw.
Informally speaking, G^* denotes the subgraph of G that contains all edges that are contained in a cycle or that connect cycles. Formally, G^* can be defined as follows.

Definition 2.1. *Let $G^0 := G$ and recursively define $G^{i+1} := G^i - (L(G_i) \cup I(G_i))$. Finally, let G^* denote the graph G^i with minimum i such that $G^i = G^{i+1}$. We will call G^* the* cyclic core *of G.*

Note that G^* is the empty graph if and only if G is acyclic (a forest). Moreover, for G being a forest of caterpillar trees, G^1 is a forest of paths. Furthermore, note that $G - V(G^*)$ is acyclic (a forest).

Definition 2.2. *For a vertex $v \in V(G^*)$ let T^v denote the tree of $G \dot{-} E(G^*)$ that is rooted at v. The tree T^v is called the* pendant tree *of v and v is called its* connection point.

Furthermore, for a vertex x in a rooted tree T let T_x denote the subtree of T rooted at x.
The following special (pendant) trees are of particular interest. A path $p = (\{v, w\}, \{w, x\})$ is called a P_2 with connection point v if $\deg(v) \geq 2$, $\deg(w) = 2$, and $\deg(x) = 1$, see Figure 2.3a for an example. Vertex w is called the middle point and we refer to it as $m(p)$ and vertex x is called the leaf of p denoted by $l(p)$. For a vertex v let $\mathcal{P}_2(v)$ denote the set of all P_2s that have v as their connection point. A Y-graph with connection point v is a subgraph consisting of the adjacent vertices v and w and two P_2s with connection point w. Optionally, w may additionally be adjacent to a leaf (see Figure 2.3b). Herein, w is called the center point of the Y-graph. We refer to w by $c(Y)$. Let $\mathcal{Y}(v)$ denote the set of all Y-graphs that have v as their connection point.



(a) A P_2 pendant (b) A Y-graph

Figure 2.3: Terminology. (a) shows a P_2 pendant with connection point v. (b) shows a Y-graph with connection point v and center point w. Here, the gray leaf may or may not be present in the Y-graph (formally, there are two different Y-graphs, one with and one without the gray leaf).

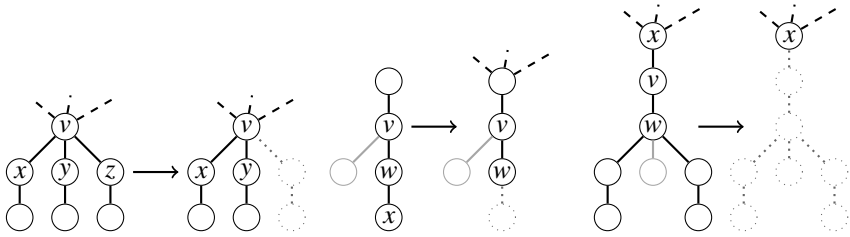
2.3 A Kernel for Two-Layer Planarization

In this section, we present a kernelization for Two-Layer Planarization parameterized by f, denoting the size of a minimum feedback edge set of the input graph. We present a number of polynomial-time executable data reduction rules and show that a graph that is reduced with respect to these rules cannot contain more than $O(f)$ vertices and edges. The kernelization consists of two phases. In the first phase, which we call "tree reduction", roughly speaking, the goal is to reduce the "acyclic part" of the input graph. In the second phase, the goal is to reduce the long non-branching paths in the remaining "cyclic core" G^*, shrinking its size to a linear function in f. We call the second phase "path reduction".

2.3.1 Tree Reduction

Subsequently, we present data reduction rules for repeatedly replacing a pendant tree T^u for some $u \in V(G^*)$ with a smaller tree, until its size is a constant value. For the explanation of the basic idea, consider a 2-claw C in T^u that has "maximal depth", that is, the sum of distances of the vertices of C to u is maximal. Clearly, all vertices that are "below" C but not contained in C are irrelevant because they are not contained in any 2-claw in T^u. Moreover, since there is no 2-claw with larger depth than C, all 2-claws that intersect C share one of the "highest" edges with C. Hence, it is optimal to destroy C by deleting one of its "highest" edges. Distinguishing all relevant cases, this observation leads to the following five data reduction rules (see Figure 2.4 for an illustration).

Tree Reduction Rule 2.1. *Let v be a vertex with $|L(v)| \geq 2$, Then, delete all but one leaf in $L(v)$.*



(a) Tree Reduction Rule 2.3 (b) Tree Reduction Rule 2.4 (c) Tree Reduction Rule 2.5

Figure 2.4: (*a*): An example of the application of Tree Reduction Rule 2.3. Note that we must delete one of the edges $\{v, x\}$, $\{v, y\}$, and $\{v, z\}$ in order to destroy the 2-claw centered at v. By symmetry, there is an optimal solution that contains the edge $\{v, z\}$. After deleting $\{v, z\}$, the dotted vertices form an isolated caterpillar and are therefore deleted. (*b*): An example for the application of Tree Reduction Rule 2.4. Note that the edge $\{w, x\}$ is not contained in any 2-claw and, hence, x can be deleted. (*c*): An example for the application of Tree Reduction Rule 2.5. Since the deletion of the edge $\{x, v\}$ destroys the same 2-claws as the deletion of any other edge in the tree rooted at x, there is an optimal solution that contains $\{x, v\}$.

Tree Reduction Rule 2.2. *Let v be a vertex with $|\mathcal{Y}(v)| \geq 1$ and $|\mathcal{Y}(v)| + |L(v)| + |\mathcal{P}_2(v)| \geq 2$. Then, for an arbitrarily chosen Y-graph $Y \in \mathcal{Y}(v)$, delete all vertices of Y except for v and decrease k by one.*

Tree Reduction Rule 2.3. *Let v be a vertex with $|\mathcal{P}_2(v)| \geq 3$ and let $\mathcal{P}_2(v) = \{p_1, p_2, \ldots, p_q\}$. Then, delete the vertices $l(p_i)$ and $m(p_i)$ for $3 \leq i \leq q$ and decrease k by $q - 2$.*

Tree Reduction Rule 2.4. *Let v be a vertex with $\widehat{\deg}_G(v) = 2$ and let p be a P_2 with $\mathcal{P}_2(v) = \{p\}$. Then, delete the vertex $l(p)$.*

Tree Reduction Rule 2.5. *Let v be a vertex with $\deg_G(v) = 2$ and let Y be a Y-graph such that $\mathcal{Y}(v) = \{Y\}$. Then, delete all vertices of Y (including v) and decrease k by one.*

The last data reduction rule for the tree reduction phase is obvious.

Tree Reduction Rule 2.6. *Let C be a connected component of G that is a caterpillar. Then, delete all vertices of C.*

Lemma 2.1. *Tree Reduction Rules 2.1–2.6 are correct. An instance reduced with respect to Tree Reduction Rules 2.1–2.6 can be computed in $O(|G|)$ time.*

Proof. First, we prove the correctness of the tree reduction rules except Tree Reduction Rules 2.1 and 2.6 (Tree Reduction Rule 2.1 is known to be correct [69], Tree Reduction Rule 2.6 is obvious), then the claimed running time.

Tree Reduction Rule 2.2 is correct. First, note that we can assume that, if a solution contains any edge from Y, then it contains the edge $e := \{v, c(Y)\}$ (since its deletion destroys also all the 2-claws that can be destroyed by any other edge in Y). We show that an optimal solution that contains e exists. Let S be an optimal solution with $e \notin S$. Clearly, this solution must contain all edges incident to v except for the edge $\{v, c(Y)\}$. Since $L(v) \neq \emptyset$ or $\mathcal{P}_2(v) \neq \emptyset$ or $|\mathcal{Y}(v)| \geq 2$, solution S must contain an edge e' that is incident to a leaf, to the middle point of a P_2, or to the center point of a Y-graph other than Y. Thus, $S \setminus \{e'\} \cup \{e\}$ clearly is a solution of same size containing e.

Tree Reduction Rule 2.3 is correct. To destroy all the 2-claws in the subgraph induced by the vertices of the P_2s in $\mathcal{P}_2(v)$ we must delete $q - 2$ edges. Since, for a $p \in \mathcal{P}_2(v)$, it is at least as good to delete edge $\{v, m(p)\}$ as to delete edge $\{m(p), l(p)\}$, by symmetry one can choose the edges $\{v, m(p_i)\}$, $3 \leq i \leq l$.

Tree Reduction Rule 2.4 is correct. Let G' denote the instance that is reduced with respect to Tree Reduction Rule 2.4. Since $\widetilde{\deg}_{G'}(v) = 1$, we know that for every solution S' for G', the vertex $m(p)$ is non-critical in $G' \doteq S'$. Consequently, all solutions for G' are also solutions for G.

Tree Reduction Rule 2.5 is correct. Let x denote the single vertex in $N(v) - c(Y)$. Note that the Y-graph Y together with x forms a graph containing a single 2-claw. Clearly, this 2-claw is best destroyed by deleting the edge $\{x, v\}$ since this also destroys all the 2-claws that can be destroyed by deleting any other edge in Y.

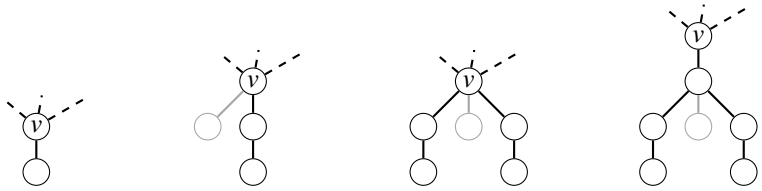
To apply the tree reduction rules efficiently, we proceed as follows. First, in a depth-first traversal of the graph, we determine the vertices in G^*. Moreover, for every vertex, we remember its parent in the spanning tree corresponding to the depth-first traversal. Then, we consider the vertices in a postorder. Doing so, for every vertex, we maintain three lists, $L(v)$, $\mathcal{P}_2(v)$, and $\mathcal{Y}(v)$. If we consider a leaf x, we add x to $L(p)$, where p is the parent of x. If we consider an inner vertex x, then we can assume that all children have already been considered because we traverse the instance bottom-up. Thus, the lists $L(x)$, $\mathcal{P}_2(x)$, and $\mathcal{Y}(x)$ have already been determined and we have all information at hand to decide whether one of the tree reduction rules can be applied to x. After applying the tree reduction rules to x, we have the following situation. If we decide that the edge from x to its parent is deleted (Tree Reduction Rules 2.4 and 2.5), then x does not have any effect on its parent. Otherwise, if neither Tree Reduction Rule 2.4 nor Tree Reduction Rule 2.5 applies to x, then we have to consider the following cases. If x is a leaf, we add x to the list L of its parent. Otherwise, if x has a single child that is a leaf, we add x to the list $\mathcal{P}_2$ of the parent of x. Note that one of these cases must apply

to x. Thus, for a vertex v, all changes can be made in $O(d(v))$ time, where $d(v)$ denotes the number of neighbors of v in $V \setminus V(G^*)$. Hence, the presented tree reduction rules can be exhaustively applied in $O(|G|) + O(\sum_{v \in V} d(v)) = O(|G|)$ time. □

The structure of an instance reduced with respect to these reduction rules is described by the following lemma, which concludes the presentation of the "tree reduction".

Lemma 2.2. *In an instance that is reduced with respect to the tree reduction rules, for each vertex $v \in V(G^*)$, its pendant tree T^v is either a singleton or isomorphic to one of the trees shown in Figure 2.5.*

Proof. To prove the lemma, we consider the height of the pendant tree T^v.
If T^v has height one, then all children of v are leaves and since the instance is reduced with respect to Tree Reduction Rule 2.1, v has exactly one child. That is, T^v is isomorphic to the tree shown in Figure 2.5a.
If T^v has height two, then, since the instance is reduced with respect to Tree Reduction Rule 2.1, every non-leaf child of v has degree two and v has at most one leaf child. Moreover, v has at most two non-leaf children since otherwise $|\mathcal{P}_2(v)| \geq 3$ and we could apply Tree Reduction Rule 2.3. Moreover, since $|L(v)| \leq 1$, the pendant tree T^v is isomorphic to one of the trees shown in Figure 2.5b and Figure 2.5c.
To prove the remaining case, we need the following observation. Let $v' \in V(T^v) - v$ be a vertex such that the height of $T^v_{v'}$ (the subtree of T^v rooted at v') is two. Since the height of T^v is neither one nor two, v' exists. We show that v' is the center of a Y-graph with connection point v. Since the height of $T^v_{v'}$ is two, it is isomorphic to either d) or e). First, note that if v' had more than two non-leaf children, then, with the same argument as in the second case above, we could apply Tree Reduction Rule 2.3. Moreover, if v' had exactly one non-leaf child, then $\widehat{\deg}_G(v') = 1$ and we could apply Tree Reduction Rule 2.4. Thus, v' has exactly two non-leaf children and at most one leaf child.
If T^v has height three, then let W denote the set of children of v such that the tree T^v_w has height two for every $w \in W$. By the above observation, we know that for every $w \in W$ the tree T^v_w together with v forms a Y-graph. Thus, $|W| = 1$, since otherwise we could apply Tree Reduction Rule 2.2. Moreover, we know that v cannot have other children, since otherwise we could apply Tree Reduction Rule 2.2. Hence, T^v is a Y-graph.
Finally, we show by contradiction that the height of T^v is at most three. Assume that the height of tree T^v is at least four and let w be some vertex in T^v such that T^v_w has height three. With the same argument as in the previous case, we



(a) single leaf (b) P_2 with optional leaf (c) two P_2s with optional leaf (d) Y-graph with optional leaf

Figure 2.5: In a reduced instance, the pendant tree T^v of a vertex $v \in V(G^*)$ is isomorphic to one of the trees shown in (a) to (d). Note that the tree shown in (d) is exactly the Y-graph as defined in Figure 2.3b.

can assume that T^v_w is a Y-graph and w has exactly one child. Then, however, the degree of w is two and we can apply Tree Reduction Rule 2.5. □

2.3.2 Path Replacement

The tree reduction rules presented in Section 2.3.1 are not sufficient to yield a problem kernel for our parameterization. For example, if the input graph G is a simple cycle, then none of the above data reduction rules applies. Recall the notions of G^* (Definition 2.1) and pendant trees. The purpose of the data reduction rules presented in this subsection is to reduce non-branching paths of G^* (hence, they are called "path reduction rules"). The first two reduction rules take care of paths containing Y-graphs as pendant trees. Then, we introduce the notion of "tokens" which allows us to handle the remaining cases in a unified manner.

In the following, we assume the input to be reduced with respect to all tree reduction rules (see previous subsection). Consider vertices $u, w \in V(G^*)$ that may be identical. We denote a path $P_{u,w} = (u = v_0, v_1, \ldots, v_\ell, v_{\ell+1} = w)$ between u and w as *degree-2 path* if $\deg_{G^*}(v_i) = 2$ for all $1 \leq i \leq l$. Its *length* is $\ell + 2$. We refer to the vertices v_i, $1 \leq i \leq l$, as *inner path vertices*. Furthermore, denote the edges $\{v_{i-1}, v_i\}$ by e_i for all $1 \leq i \leq \ell + 1$. Throughout this subsection, let P be some degree-2 path in the given graph G and let v_i be some inner path vertex of P. In this context, let $T_P := G[\bigcup_{i=1}^{l} V(T^{v_i}) \cup \{u, w\}]$ and for $1 \leq i \leq j \leq l$, let $T_{i,j}$ denote the subtree of T_P containing all vertices reachable from v_i in $T_P \dot{-} \{e_i, e_{j+1}\}$. Note that $T_{i,i} = T^{v_i}$.

The first two path reduction rules handle all Y-graphs that have a vertex on a degree-2 path as their connection point. For their presentation, we need two

additional tools. First, observe that for any Y-graph Y with connection point v, deleting the edge of Y that is incident to v is at least as good as deleting any combination of edges of Y. This allows us to assume optimal solutions to contain either this edge or all other edges incident to v. Furthermore, since pendant trees do not overlap, we can assume this for all vertices v.

Observation 2.1. *There is an optimal solution S for G such that for each Y-graph Y with connection point v, it holds that $S \cap E(Y) \subseteq \{\{v, c(Y)\}\}$.*

Second, observe that we can assume certain vertices to be non-critical in the target caterpillar-forest corresponding to an optimal solution.

Lemma 2.3. *Let v be a degree-2 vertex in G^* such that T^v is neither a singleton nor a Y-graph. Then, there is an optimal solution S for G such that v is non-critical in $G \dot{-} S$.*

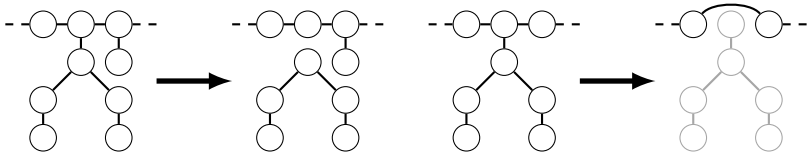
Proof. Let S denote an optimal solution for G such that v is critical in $G \dot{-} S$. We show that there is a solution S' with $|S'| = |S|$ such that v is non-critical in $G \dot{-} S'$. Since v is critical in $G \dot{-} S$, by definition, v is a leaf in $G \dot{-} S$ and adjacent to a vertex x with non-leaf degree two in $G \dot{-} S$. Since T^v is not a Y-graph, Lemma 2.2 implies that every vertex in $V(T^v)$ (except for v) has non-leaf degree at most one in G and, hence, also in $G \dot{-} S$. This implies that $x \notin V(T^v)$. Let $F := E(G) \cap \{\{v, z\} : z \in V(T^v)\}$. Since v is a leaf adjacent to x in $G \dot{-} S$, we have $F \subseteq S$ and since T^v is not a singleton, we know that $|F| \geq 1$. Observe that deleting $\{v, x\}$ from $G \dot{-} S$ renders v isolated. Thus, subsequently undoing the edge deletions in F does not create a cycle or 2-claw. Hence, $S' := (S \setminus F) \cup \{\{v, x\}\}$ is also a solution for G. Finally, note that $G \dot{-} S'$ contains T^v as a connected component. Since T^v is not a Y-graph, v is non-critical in $G \dot{-} S'$. □

In the proof of Lemma 2.3, the only pendent tree affected by the modifications in S is T^v, which, by assumption, is not a Y-graph. Hence, we can apply Lemma 2.3 and Observation 2.1 independently. We use such an optimal solution to prove the correctness of the next reduction rule.
The first path reduction rule describes Y-graphs Y with connection point v_i for which it is optimal to delete the edge $\{v_i, c(Y)\}$ (see also Figure 2.6).

Path Reduction Rule 2.1. *Let P denote a degree-2 path and let both v_i and v_{i+1} be inner path vertices of P such that $Y := T^{v_i}$ is a Y-graph. If*

1. *$T^{v_{i+1}}$ is neither a singleton nor a Y-graph, or*
2. *v_{i+2} is an inner path vertex, $\deg_G(v_{i+1}) = 2$, and $T^{v_{i+2}}$ is either a singleton or a Y-graph,*

then delete $\{v_i, c(Y)\}$ and decrease k by one.



(a) application of Case 1 of Path Reduction Rule 2.1

(b) application of Case 2 of Path Reduction Rule 2.2

Figure 2.6: Path Reduction Rules 2.1 and 2.2. Gray subgraphs are deleted by the reduction rules.

Lemma 2.4. *Path Reduction Rule 2.1 is correct.*

Proof. For the correctness, we show that if Path Reduction Rule 2.1 applies to v_i, then there is an optimal solution for G that contains $\{v_i, c(Y)\}$ and, hence, it is safe to remove edge $\{v_i, c(Y)\}$. In the following, let $e_Y := \{v_i, c(Y)\}$ and let S denote an optimal solution for G as described by Observation 2.1 and Lemma 2.3. That is, S is an optimal solution such that

1. for each Y-graph Y' with connection point v', $S \cap E(Y') \subseteq \{\{v', c(Y')\}\}$, and

2. if $T^{v_{i+1}}$ is neither a singleton nor a Y-graph, then v_{i+1} is non-critical in $G \doteq S$.

As discussed above, such a solution exists.
If S contains e_Y, then we are done, hence, in the following, we assume that $e_Y \notin S$. Then, by Property 1, S does not contain any edge of Y, implying $e_i, e_{i+1} \in S$. If v_{i+1} is non-critical in $G \doteq S$, then $S \setminus \{e_{i+1}\} \cup \{e_Y\}$ is an optimal solution for G containing e_Y. To show that v_{i+1} is non-critical in $G \doteq S$, assume that v_{i+1} is critical in $G \doteq S$. In Case 1 of Path Reduction Rule 2.1, this contradicts Property 2. In Case 2 of Path Reduction Rule 2.1, $\deg_G(v_{i+1}) = 2$ and $T^{v_{i+2}}$ is either a singleton or a Y-graph. Since $e_{i+1} \in S$, the fact that v_{i+1} is critical in $G \doteq S$ implies that v_{i+2} is adjacent to v_{i+1} in $G \doteq S$ and $\widehat{\deg}_{G \doteq S}(v_{i+2}) \geq 2$. If $T^{v_{i+2}}$ is a Y-graph, then, since S does not contain $\{v_{i+1}, v_{i+2}\}$, Property 1 implies that S contains the edge $\{v_{i+2}, c(T^{v_{i+2}})\}$. Hence, v_{i+2} has at most two neighbors in $G \doteq S$, regardless of whether $T^{v_{i+2}}$ is a Y-graph or a singleton. However, since one of these neighbors is v_{i+1} (a leaf), v_{i+2} has non-leaf degree at most one in $G \doteq S$, a contradiction to v_{i+1} being critical in $G \doteq S$. □

The second data reduction rule handles almost all remaining cases where a Y-graph occurs as the pendant tree of some inner path vertex v_i of P by "bypassing" v_i in P.

Path Reduction Rule 2.2. *Let P denote a degree-2 path and let both v_i and v_{i+1} be inner path vertices of P such that $Y := T^{v_i}$ is a Y-graph. If*

1. *$T^{v_{i+1}}$ is a Y-graph, or*
2. *v_{i-1} and v_{i+1} have degree two, or*
3. *v_{i+2} is an inner path vertex, $\deg(v_{i+1}) = 2$, and $T^{v_{i+2}}$ is neither a singleton nor a Y-graph,*

then remove all vertices of Y from G, insert the edge $e = \{v_{i-1}, v_{i+1}\}$, and decrease k by one.

Lemma 2.5. *Path Reduction Rule 2.2 is correct.*

Proof. Let G' denote the graph that results from applying Path Reduction Rule 2.2 to some v_i in G and let $e_Y := \{v_i, c(Y)\}$. Let $\hat{G}$ denote the result of contracting e_{i+1} in $G \div e_Y$ and observe that $\hat{G}$ is identical to G' with the exception of one connected component (containing all vertices of Y but v_i) which is a caterpillar. Consequently, $\hat{G}$ and G' are equivalent in the sense that a solution for one is also a solution for the other (considering that e_i in $\hat{G}$ plays the role of e in G'). In the following, we show that G has a solution S of size at most k if and only if G' has a solution S' of size at most $k - 1$.

"⇒:" By Observation 2.1, either e_Y or both e_i and e_{i+1} are in S. If $e_Y \in S$, then, since contracting an edge does not create 2-claws or cycles, $S' := S \setminus \{e_Y\}$ is a solution of size at most $k - 1$ for $\hat{G}$ and, thus, for G'. Otherwise, $e_i, e_{i+1} \in S$. However, by construction, $G' \div e$ is a subgraph of $G \div \{e_i, e_{i+1}\}$ and thus, $S' := (S \setminus \{e_i, e_{i+1}\}) \cup \{e\}$ is a solution for G' of size at most $k - 1$.

"⇐:" If a solution S' for G' of size at most $k - 1$ contains e, then the equivalent solution $\hat{S}$ for $\hat{G}$ contains e_i, and clearly, $S := \hat{S} \cup \{e_{i+1}\}$ is a solution of size at most k for G. Thus, in the following, we assume that there is no solution for G' of size at most $k - 1$ that contains e (in particular, $e \notin S'$ and, thus, v_{i-1} and v_{i+1} are neighbors in $G' \div S'$).

Observe that the subdivision of an edge e' of a caterpillar can only create a 2-claw if e' is incident to a critical vertex. Hence, if v_{i-1} and v_{i+1} are both non-critical in $G' \div S'$, then we can subdivide e without affecting the solution and thus, $S := S' \cup \{e_Y\}$ is a solution of size at most k for G. Hence, in the following, we assume that v_{i-1} or v_{i+1} is critical in $G' \div S'$. Since v_{i-1} and v_{i+1} are adjacent in $G' \div S'$, this implies that one of them has degree at least three in $G' \div S'$ while the other is a leaf in $G' \div S'$. In the following, we consider the three cases of Path Reduction Rule 2.2 separately.

Case 1: The first condition of Path Reduction Rule 2.2 applies.

Then, $Y' := T^{v_{i+1}}$ is a Y-graph. Let $e_{Y'} := \{v_{i+1}, c(Y')\}$. By Observation 2.1, we can assume that S' contains either $e_{Y'}$ or both e and e_{i+2} (if S' contains $e_{Y'}$ and e but not e_{i+2}, then we can exchange $e_{Y'}$ for e_{i+2} in S'). However, by the assumption

that $e \notin S'$, it follows that $e_{i+2} \notin S'$ and $e_{Y'} \in S'$, implying $\deg_{G' \dot{-} S'}(v_{i+1}) = 2$. Thus, neither v_{i+1}, nor v_{i-1} is critical in $G' \dot{-} S'$, a contradiction.
Case 2: The second condition of Path Reduction Rule 2.2 applies.
Then, $\deg_G(v_{i-1}) = \deg_G(v_{i+1}) = 2$, implying that neither v_{i-1} nor v_{i+1} has degree at least three in $G' \dot{-} S'$, contradicting the assumption that v_{i-1} or v_{i+1} is critical.
Case 3: The third condition of Path Reduction Rule 2.2 applies.
By Lemma 2.3 we can assume that v_{i+2} is non-critical in $G' \dot{-} S'$. Clearly, v_{i-1} is non-critical in $G' \dot{-} S'$ since $\deg_G(v_{i+1}) = 2$. Hence, v_{i+1} is critical in $G' \dot{-} S'$ and, thus, $S' \cup \{e\}$ isolates v_{i+1}. Since v_{i+2} is non-critical in $G' \dot{-} S'$, it follows that $(S' \cup \{e\}) \dot{-} e_{i+2}$ is a solution of size at most k for G' containing e, a contradiction. □

The two path reduction rules presented so far eliminate Y-graphs in all long degree-2 paths. In the following, consider P to be *Y-graph-free*, that is, P does not contain an inner path vertex whose pendant tree is a Y-graph. Consider a graph that is reduced with respect to Path Reduction Rules 2.1 and 2.2. All degree-2 paths P' that are not Y-graph-free contain at most two inner path vertices, whose pendant trees are a singleton and a Y-graph, respectively. Thus, it is clear that $|V(T_{P'})| \leq 10$. In the following, we focus on Y-graph-free degree-2 paths. In this case, we can show that we do not need to consider deleting edges in pendant trees, thus allowing us to restrict our attention to deleting edges on the degree-2 path.

Lemma 2.6. *Let G be reduced with respect to the tree reduction rules. Then, there is an optimal solution S for G that does not contain edges of the pendant tree T^v of any $v \in V(G^*)$ unless T^v is a Y-graph.*

Proof. Let S denote an optimal solution for G that contains an edge e of some T^v. Without loss of generality, e is incident to v. Since S is optimal, there is a 2-claw Z in $G \dot{-} (S \dot{-} e)$. If Z is centered at v, then there are exactly two non-leaf neighbors u, u' of v in $G \dot{-} S$. By Lemma 2.2, u or u' is not in T^v. Without loss of generality, u is not in T^v. Then, however, $S \dot{-} \{e, \{u, v\}\}$ is an optimal solution for G avoiding T^v.
If Z is not centered at v, then let $w \in N_G(v)$ denote the center of Z. Since T^v is not a Y-graph, w is not in T^v. Since S is a solution for G, we know that $N_{G \dot{-} S}(v) = \{w\}$. Hence, $S \setminus \{e, \{v, w\}\}$ is an optimal solution for G. □

Corollary 2.1. *Let P be a Y-graph-free degree-2 path. Then, there is an optimal solution for G that does not contain any edge of $E(T_P) \setminus E(P)$.*

In order to handle the remaining degree-2 paths (recall the definition on page 33) in a unified manner, we introduce the notion of "tokens" as sets of consecutive edges on the degree-2 paths. Consider a vertex v on a degree-2 path P that is

the center of a 2-claw. Then, Corollary 2.1 tells us that this 2-claw must be destroyed by deleting an edge of P. We model this fact by letting v generate a token containing all edges that may be deleted in order to destroy this 2-claw. The introduction of this notion is split into three parts. First, we define tokens, second, we point out how they are generated, and third, we specify what it means to destroy a token.

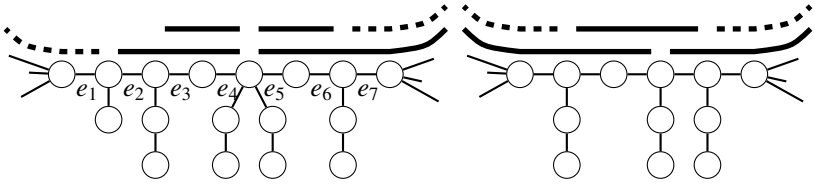
In the following, a vertex v in P is called *crossable* if $\deg_G(v) = 2$. A *token* K of P is a set of at most four consecutive edges of P. Let v_i be a vertex in P. If $i \leq \ell - 1$, then the *upper token* $K^{\text{up}}(v_i)$ of v_i is $\{e_{i+1}, e_{i+2}\}$ if v_{i+1} is crossable and it is $\{e_{i+1}\}$, otherwise. Equivalently, if $i \geq 2$, then the *lower token* $K^{\text{low}}(v_i)$ of v_i is $\{e_{i-1}, e_i\}$ if v_{i-1} is crossable and it is $\{e_i\}$ otherwise. In this sense, we say that tokens can only "span" over crossable vertices. For the vertices u, v_1, v_ℓ, and w, we need the following auxiliary tokens: $K^{\text{low}}(u) := \{\diamond^{\text{low}}\}$, $K^{\text{up}}(w) := \{\diamond^{\text{up}}\}$, $K^{\text{low}}(v_1) := \{\diamond^{\text{low}}, e_1\}$, and $K^{\text{up}}(v_\ell) := \{e_{\ell+1}, \diamond^{\text{up}}\}$.

Each inner path vertex v_i of P for which $\mathcal{P}_2(v_i) \neq \emptyset$ *generates* tokens in the following way: If $|\mathcal{P}_2(v_i)| = 1$ (in this case T^{v_i} is isomorphic to one of the trees shown in Figure 2.5b), then v_i generates one token $K^{\text{up}}(v_i) \cup K^{\text{low}}(v_i)$. If $|\mathcal{P}_2(v_i)| = 2$ (in this case T^{v_i} is isomorphic to one of the trees shown in Figure 2.5c), then v_i generates two tokens, $K^{\text{up}}(v_i)$ and $K^{\text{low}}(v_i)$. We define $\mathcal{K}(v_i)$ as the set of tokens generated by v_i and $\mathcal{K}(P)$ as the set of all tokens generated by inner path vertices of P. See Figure 2.7 for an illustration. We can observe that only vertices that are centers of 2-claws generate tokens. Moreover, all 2-claws centered at inner path vertices are represented in the tokens of $\mathcal{K}(P)$, implying that all 2-claws on P can be destroyed by removing an edge of each token in $\mathcal{K}(P)$.

Observation 2.2. *For each $K \in \mathcal{K}(P)$, there is a 2-claw C centered at the vertex that generates K such that $E(C) \cap E(P) = K$ Furthermore, for each 2-claw C that is centered at an inner path vertex of P, there is a token $K \in \mathcal{K}(P)$ such that K is a subset of all edges that are in both C and P.*

A non-auxiliary token (token that does not contain $\diamond^{\text{up}}$ or $\diamond^{\text{low}}$) $K \in \mathcal{K}(P)$ is *destroyed* by an edge set E' if $K \cap E' \neq \emptyset$. Observe that if K contains $\diamond^{\text{up}}$ or $\diamond^{\text{low}}$, then it contains e_1 or $e_{\ell+1}$, respectively. We say that a token containing e_1 and $\diamond^{\text{low}}$ is destroyed by E' if either $K \cap E' \neq \emptyset$ or E' contains all edges incident to v_0 except for e_1. An auxiliary token containing $e_{\ell+1}$ is destroyed analogously. Informally speaking, a token represents the need to delete an edge. By Corollary 2.1 it suffices to consider edges in P. Thus, the task is to destroy all tokens by deleting only few edges.

The following path reduction rules are designed to shrink degree-2 paths and rely heavily on the notion of tokens. The first data reduction rule reduces degree-2 paths that do not contain any tokens.



(a) Illustration of token-separators. The tokens are $\{e_2, e_3, e_4\}$, $\{e_3, e_4\}$, $\{e_5, e_6\}$, and $\{e_5, e_6, e_7, \diamond^{\text{up}}\}$, the separators are v_1 and v_4.

(b) An example of a chained degree-2 path.

Figure 2.7: Illustration of the tokens generated by degree-2 paths. Horizontal bars depict tokens, dashed lines represent auxiliary tokens. Bars are bent if they contain $\diamond^{\text{low}}$ or $\diamond^{\text{up}}$.

Path Reduction Rule 2.3. *If there is a degree-2 path P with $|V(T_P)| > 7$ and $\mathcal{K}(P) = \emptyset$, then contract $T_{2,\ell-1}$ to a single vertex.*

Lemma 2.7. *Path Reduction Rule 2.3 is correct.*

Proof. Let G denote the input graph and G' the graph that results from applying Path Reduction Rule 2.3 to a degree-2 path P in G. We show that for each solution S for G there is also a solution for G' of the same size, and vice versa.
By Observation 2.2, $\mathcal{K}(P) = \emptyset$ implies that there is no 2-claw centered at an inner path vertex of P. However, no other 2-claw can intersect $T_{2,\ell-1}$ and, thus, the set of edges that are in 2-claws in G and in G' are equal, implying that $G \doteq S$ contains 2-claws if and only if $G' \doteq S$ does.
Observe that $T_{2,\ell-1}$ is a tree and edge contraction cannot create cycles. Hence, if S does not contain any edge from $T_{2,\ell-1}$, then S also breaks all cycles of G'. Otherwise, there is an edge e in S and in $T_{2,\ell-1}$ and we can exchange e for e_0 in S, thereby obtaining a solution S' for G'.
Finally, observe that $|V(T_P)| > 7$ and $\mathcal{K}(P) \neq \emptyset$ implies $|V(T_{2,\ell-1})| > 1$. □

Next, we focus on degree-2 paths generating tokens. Note that the end vertices u and w of P could be centers of 2-claws containing inner path vertices of P. To account for this possibility, we define $\mathcal{K}'(P) := \mathcal{K}(P) \cup \{K^{\text{up}}(u), K^{\text{low}}(w)\}$. Note also that tokens are basically edge sets, which may overlap. This behavior is exploited in the following reduction rules requiring a more formal definition. We call an inner path vertex v_i of P a *token separator* if there is no token in $\mathcal{K}'(P)$ containing both e_i and e_{i+1}. If P does not have a token-separator, then P is called *chained* (see Figure 2.7b).

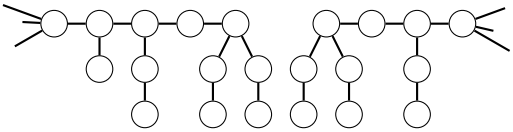


Figure 2.8: The graph that results from applying Path Reduction Rule 2.4 to the graph shown in Figure 2.7a.

In the following, we present a reduction rule that works on degree-2 paths P containing a token separator v. In particular, this reduction rule splits P at v, duplicating T^v. This operation leaves all tokens intact, indicating the correctness of the reduction rule (see Figure 2.8 for an illustration).

Path Reduction Rule 2.4. *Let P be a degree-2 path with $\mathcal{K}(P) \neq \emptyset$ and P is not chained. Let v_i be a token separator of P. Then, replace its pendant tree T^{v_i} by two copies of T^{v_i}, connect one to v_{i-1} (by inserting an edge between its connection point and v_{i-1}), and connect the other to v_{i+1}.*

Lemma 2.8. *Path Reduction Rule 2.4 is correct.*

Proof. Let G' denote the graph that results from applying Path Reduction Rule 2.4 to P. In the following, we show for all k that (G, k) is a yes-instance if and only if (G', k) is a yes-instance.
"⇒": By Corollary 2.1, we know that there is a solution S of size at most k for G that does not delete edges in T^{v_i}. For the sake of contradiction, assume that S is not a solution for G'. Since all cycles in G' are cycles in G, there is a 2-claw C in $G' \dot{-} S$ that is not in $G \dot{-} S$. If C contains both copies of v_i, then there is a cycle in G that consists entirely of edges of C. Since S is a solution for G, it is clear that S breaks this cycle, contradicting C being a 2-claw in $G' \dot{-} S$. If C contains only one of the copies of v_i, then we can delete the other copy from $G' \dot{-} S$ without destroying C. The resulting graph is a subgraph of $G \dot{-} S$ that contains C, contradicting S being a solution for G.
"⇐": Let S denote a solution for G' containing at most k edges. If S contains any edge e of the copy of T^{v_i} connected to v_{i-1} (see Path Reduction Rule 2.4), then it is easy to see that $(S \dot{-} e) \cup \{e_i\}$ is also a solution for G' that does not contain such an edge. Analogously, we can assume that S does not contain an edge of the copy of T^{v_i} connected to v_{i+1}. In the following, we show that S is also a solution for G. For the sake of contradiction, assume that $G \dot{-} S$ is not a forest of caterpillars. First, assume that there is a 2-claw C in $G \dot{-} S$. Note that both e_i and e_{i+1} are contained in C, since otherwise, C is entirely contained in G' implying that C is also a 2-claw in $G' \dot{-} S$.

Case 1: The center of C is v_i. Then, $|\mathcal{P}_2(v_i)| \geq 1$ and, thus, $\mathcal{K}(v_i) \neq \emptyset$. Since v_i is a token separator, $|\mathcal{K}(v_i)| = 2$ and, thus, $|\mathcal{P}_2(v_i)| = 2$. However, since both e_i and e_{i+1} are in C, we know that C only uses one of the two P_2s in $\mathcal{P}_2(v_i)$. Since S does not contain any edge of any copy of T^{v_i}, we know that there is another 2-claw C' in $G \dot{-} S$ that contains both P_2s of $\mathcal{P}_2(v_i)$ and only one of e_i and e_{i+1}. As observed above, C' is entirely contained in $G' \dot{-} S$, contradicting S being a solution for G'.
Case 2: The center of C is not v_i. Then, since e_i and e_{i+1} are in C, we know that C is centered either at v_{i-1} or v_{i+1}. Without loss of generality, let v_{i+1} be the center of C. If v_i is crossable in P, then v_{i+1} generates a token that spans over v_i, contradicting v_i being a token-separator of P. Hence, v_i is not crossable and, consequently, its degree in G is at least three, implying that T^{v_i} is not a singleton. Thus, there is also a 2-claw C' in G that uses an edge of T^{v_i} instead of e_i. Since C' is entirely contained in G', the solution S contains an edge of C' and by the assumption that S does not contain an edge of any copy of T^{v_i}, we know that S contains an edge of C' that is also in C, contradicting the existence of C in $G \dot{-} S$.
Second, assume that there is a cycle in $G \dot{-} S$. Then, it contains v_i, since otherwise, it is also a cycle in $G' \dot{-} S$. By definition, P is a subpath of this cycle. However, since $\mathcal{K}(P) \neq \emptyset$, Observation 2.2 implies that there is a 2-claw C in G that is centered at a vertex of P. By the argument above, S contains an edge e' of C. If e' is not in P, then for one of the endpoints of P, the solution S contains all but one of the edges incident to it (this is represented by the $\diamond^{\text{up}}$ and $\diamond^{\text{low}}$ symbols). Otherwise, e' is in P. Both cases contradict the existence of a cycle with subpath P. □

Having dealt with degree-2 paths that contain token separators or no tokens at all, only chained degree-2 paths remain untouched. In the following, we show that, if P is such a chained degree-2 path, then P can be reduced. To this end, we first observe that each edge deletion can only destroy two tokens. This easily follows from the fact that being crossable and generating tokens is mutually exclusive.

Observation 2.3. *Let P be a degree-2 path. Then, each edge of P is contained in at most two tokens.*

Next, we use this observation to "contract" P such that the outermost tokens remain the same (see Figure 2.9). To this end, recall that $T_{i,j}$ is the subtree of T_P containing all vertices reachable from v_i in $T_P - \{e_i, e_{j+1}\}$. Furthermore, find an example of a chained degree-2 path in Figure 2.7b.

Path Reduction Rule 2.5. *Let P be a chained degree-2 path with $|\mathcal{K}(P)| \geq 3$. Let $M_P := \{i \in \mathbb{N} : \mathcal{K}(v_i) \neq \emptyset\}$, let $p := \min M_P$ and $q := \max M_P$. If $|\mathcal{K}(P)|$ is even, then delete $T_{p+1,q-1}$, insert the edge $e := \{v_p, v_q\}$, and reduce k by $(|\mathcal{K}(P)| - 2)/2$; otherwise delete $T_{p+1,q}$, insert the edge $e := \{v_p, v_{q+1}\}$, and reduce k by $(|\mathcal{K}(P)| - 1)/2$.*

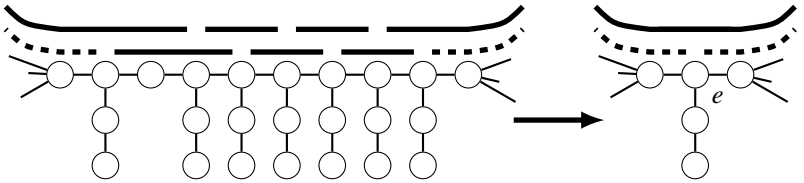


Figure 2.9: An application of Path Reduction Rule 2.5 with $|\mathcal{K}(P)| = 7$. In the example, $M_p = \{1, 3, 4, 5, 6, 7, 8\}$ and, hence, $p = 1$, $q = 8$, and $e = \{v_1, v_9\}$.

Lemma 2.9. *Path Reduction Rule 2.5 is correct.*

Proof. First, consider the case that $|\mathcal{K}(P)|$ is even. Let G' denote the graph that results from applying Path Reduction Rule 2.5 to P, let $k' := k - (|\mathcal{K}(P)| - 2)/2$, and let $E_T := \{e_{p+1}, e_{p+2}, \ldots, e_q\}$. Since neither v_p nor v_q is crossable, we know that $|\mathcal{K}(P)| - 2$ out of all tokens of P are subsets of E_T. We show that (G, k) is a yes-instance if and only if (G', k') is a yes-instance.

"⇒": Let S be a solution of size at most k for G, let $S_T := S \cap E_T$, and let $S' := S \setminus E_T$. Since S destroys all tokens in $\mathcal{K}(P)$ and, by Observation 2.3, each edge in S can destroy at most two tokens of $\mathcal{K}(P)$, we know that $|S_T| \geq (|\mathcal{K}(P)| - 2)/2$. If S' is a solution for G', then we are done; otherwise, $G' \dot{-} S'$ contains a cycle or a 2-claw. Assume that $G' \dot{-} S'$ contains a cycle. Then, since $G \dot{-} S$ is acyclic, this cycle contains all edges of E_T. If $|S_T| > (|\mathcal{K}(P)| - 2)/2$, then, by definition, $|S'| < k'$ and, hence, $S' \cup \{e\}$ is a solution of size at most k' for G' (note that $G' \dot{-} e$ is a subgraph of G). If $|S_T| = (|\mathcal{K}(P)| - 2)/2$, then S_T cannot destroy all tokens in $\mathcal{K}(P)$. Hence, there are some edges in S' that destroy $\mathcal{K}(v_p)$ or $\mathcal{K}(v_q)$. It is easy to see that at least one of these edges is on the cycle in $G' \dot{-} S'$, which is a contradiction.

In the following, we assume that $G' \dot{-} S'$ is acyclic but contains a 2-claw C. Observe that C contains e, since otherwise, C is also in $G \dot{-} S$. If C is not centered at an inner path vertex of P, then, since v_p and v_q both generate tokens (and therefore, have degree at least three in G), Corollary 2.1 implies that S' contains an edge of C. Hence, C is centered at v_p or v_q. Without loss of generality, assume that C is centered at v_p. Let e_i denote an edge in $S \cap K(v_p)$ (which, by Observation 2.2, exists). Since $e_i \notin S'$, we know that $e_i \in S_T \cap K(v_p)$. Then, however, S_T destroys at least $|\mathcal{K}(P)| - 1$ tokens of $\mathcal{K}(P)$. Since $|\mathcal{K}(P)|$ is even, $|S_T| \geq |\mathcal{K}(P)|/2$ and, thus, $|S'| < k'$. As a consequence, $S' \cup \{e\}$ is a solution for G' containing at most k' edges.

"⇐": Let S' be a solution of size at most k' for G' and recall that, by Observation 2.2, destroying all tokens in $\mathcal{K}(P)$ destroys all 2-claws centered at inner path vertices. If $e \notin S'$, then it suffices to show that we can extend S' to a solution S

for G by adding $k - k' = (|\mathcal{K}(P)| - 2)/2$ edges of E_T that destroy $|\mathcal{K}(P)| - 2$ tokens. Since P is chained, this is possible. If $e \in S'$, then we extend $S' \dot{-} e$ to a solution S for G by adding $|\mathcal{K}(P)|/2$ edges of E_T that destroy all tokens in $\mathcal{K}(P)$. Again, this is possible because P is chained.

Next, consider the case that $|\mathcal{K}(P)|$ is odd. Let G' denote the graph that results from applying Path Reduction Rule 2.5 to P, let $k' := k - (|\mathcal{K}(P)| - 1)/2$, and let $E_T := \{e_{p+1}, e_{p+2}, \ldots, e_{q+1}\}$. Furthermore, observe that at least $|\mathcal{K}(P)| - 2$ of the tokens of P are subsets of E_T. We show that (G, k) is a yes-instace if and only if (G', k') is a yes-instance.

"⇒": Let S denote a solution of size at most k for G, let $S_T := S \cap E_T$, and let $S' := S \setminus E_T$. By the same arguments as in the case that $|\mathcal{K}(P)|$ is even, it follows that $|S_T| \geq (|\mathcal{K}(P)| - 2)/2$ and that $G' \dot{-} S'$ is acyclic, but contains a 2-claw C that contains e. Clearly, by definition of p and q, if C is not centered at v_p, then the center of C is not in P. Then, however, S_T destroys $|\mathcal{K}(P)| + 1$ tokens and, by Observation 2.3, S_T contains at least $(|\mathcal{K}(P)| + 1)/2$ edges, implying $|S'| < k'$. Hence, $S' \cup \{e\}$ is a solution of size at most k' for G'. In the following, we assume that C is centered at v_p and consider the token $\mathcal{K}(v_q)$ that is destroyed by S.

Case 1: $(K^{\text{low}}(v_q) \cup \{e_{q+1}\}) \cap S \neq \emptyset$. Since $K^{\text{low}}(v_q) \cup \{e_{q+1}\} \subseteq E_T$, this means that S_T destroys *all* tokens in $\mathcal{K}(P)$. By Observation 2.3, $|S_T| \geq |\mathcal{K}(P)|/2$. Since $\mathcal{K}(P)$ is odd, $|S_T| \geq (|\mathcal{K}(P)| + 1)/2$ and, thus, $S' \cup \{e\}$ is a solution for G' containing at most k' edges.

Case 2: $(K^{\text{low}}(v_q) \cup \{e_{q+1}\}) \cap S = \emptyset$. Since v_{q+1} is not a token-separator, v_{q+1} is either the end-vertex of P or a degree-two vertex. Then, however, S contains all edges incident to v_{q+1} except for e_q. Since $S \setminus E_T$ also contains all these edges, it is clear that S' contains them and, thus, C is destroyed by S', a contradiction to the assumption above.

"⇐": Let S' denote a solution of size at most k' for G'. If $e \notin S'$, then it suffices to show that we can extend S' to a solution S for G by adding $k - k' = (|\mathcal{K}(P)| - 1)/2$ edges of E_T that destroy $|\mathcal{K}(P)| - 1$ tokens. Since P is chained, this is possible. If $e \in S'$, then we can analogously extend $(S' \dot{-} e) \cup \{e_{q+1}\}$ to a solution S for G. □

With the presented tree- and path reduction rules, we can now upper-bound the size of the graph that remains after all reduction rules have been applied exhaustively by a function of the size of a minimum feedback edge set. To this end, we first show that the number of vertices and edges in each degree-2 path and its pendant trees is small.

Lemma 2.10. *Let G be reduced with respect to all path reduction rules and let V_3 denote the set of vertices with degree at least 3 in G^*. If two vertices $u, w \in V_3$ are connected in G by a degree-2 path P, then $|V(T_P) \setminus \{u, w\}| \leq 10$ and $|E(T_P)| \leq 11$.*

Proof. Let P denote the degree-2 path between u and w.
Case 1: There is some vertex v_i in P such that its pendant tree T^{v_i} is a Y-graph. Then, since G is reduced with respect to Path Reduction Rules 2.1 and 2.2, P contains at most two inner path vertices, one of which has degree 2. All in all, $|V(T_P) \setminus \{u, w\}| \leq 8$ and $|E(T_P)| \leq 9$.
Case 2: $\mathcal{K}(P) = \emptyset$. Since G is reduced with respect to Path Reduction Rule 2.3, we know that $|V(T_P) \setminus \{u, w\}| \leq 5$ and, hence, $|E(T_P)| \leq 6$.
Case 3: $\mathcal{K}(P) \neq \emptyset$. Since G is reduced with respect to Path Reduction Rule 2.4, P is chained. But G is also reduced with respect to Path Reduction Rule 2.5, and, thus, P contains at most six inner path vertices. At most two of these inner path vertices generate tokens, all others are crossable (and therefore, have degree two). Let v_p and v_q denote these two vertices in P. Then, their pendant trees T^{v_p} and T^{v_q} are P_2s. Thus, $T_P \setminus \{u, w\}$ contains at most 10 vertices and at most 11 edges. □

With the help of Lemma 2.10, we can now upper-bound the total number of vertices and edges in graphs that are reduced with respect to all presented data reduction rules.

Theorem 2.1. Two-Layer Planarization *admits a problem kernel containing at most* $44(f-1)$ *vertices and at most* $45(f-1)$ *edges, with* $f > 0$ *being the size of a minimum feedback edge set of* G. *The problem kernel can be constructed in* $O((f+1) \cdot |G|)$ *time.*

Proof. Assume that the input G is reduced with respect to all presented data reduction rules. For the analysis of the kernel size, we need the following notation. Let V_3 denote the set of vertices with degree at least 3 in G^* and let $G_3^* := (V_3, E_3)$ denote the multigraph on V_3 that contains an edge for every maximal degree-2 path in G. More specifically, E_3 contains an edge $\{u, w\}$ for every edge $\{u, w\} \in E$ with $u, w \in V_3$, and, in addition, E_3 contains an edge $\{u, w\}$ for every maximal degree-2 path of length at least three between two (not necessarily different) vertices $u, w \in V_3$. Thus, G_3^* may contain loops. Furthermore, let F with $|F| = f$ be a minimum feedback edge set of G and let F_3 be a minimum feedback edge set of G_3^* (we require that a feedback edge set of G_3^* contains all loops and all but at most one edge between any two vertices). Clearly, $|F_3| \leq f$ and $G_3^* \dot{-} F_3$ is a forest and, thus, $|E_3| \leq |V_3| + f - 1$. Since the minimum degree[15] of a vertex in G_3^* is 3, we know that $\sum_{v \in V_3} \deg_{G_3^*}(v) \geq 3|V_3|$, and since the sum on the left hand side equals $2|E_3|$, we know that $2(|V_3| + f - 1) \geq 3|V_3|$, implying $|V_3| \leq 2(f-1)$ and $|E_3| \leq 3(f-1)$.
With the size of G_3^* bounded, we can use Lemma 2.10 to bound the size of G. Each edge in G_3^* corresponds to a degree-2 path in G^*. Each vertex in V_3 may

[15] A loop at vertex v contributes 2 to the degree of v.

additionally be incident to a pendant tree (see Figure 2.5). Thus, we can bound the number of vertices in G by $|V(G)| \leq |E_3| \cdot 10 + |V_3| \cdot 6 + |V_3| \leq 44(f-1)$ and the number of edges in G by $|E(G)| \leq 45(f-1)$.
It remains to show the running time of the kernelization. In the following, we prove that the path reduction rules can be applied to G in $O((f+1) \cdot |G|)$ time. Recall that, in the proof of Lemma 2.1, $L(v)$, $\mathcal{P}_2(v)$ and $\mathcal{Y}(v)$ are constructed in linear time for all vertices of G. Since both Path Reduction Rule 2.1 and 2.2 reduce the number of edges, they can only apply $|E|$ times in total. Each such application can be performed in constant time. It is easy to see that no degree-2 path is subject to more than one of the Path Reduction Rules 2.3–2.5. In the worst case, the application of such a path reduction rule requires reapplying the tree reduction rules, which may take $O(|G|)$ time. Since $|E(G_3^*)| \leq O(f)$, we can bound the running time by $O((f+1) \cdot |G|)$. □

2.4 Search Tree Algorithms

In this section, we provide two algorithms solving the Two-Layer Planarization problem in $O(6^f \cdot f^2 + (f+1) \cdot |G|)$ time and $O(3.8^f \cdot f + (f+1) \cdot |G|)$ time, respectively. Both algorithms employ a search-tree traversal, branching on different structures. The first algorithm is an adaptation of the search-tree algorithm of Dujmović et al. [69], the second is an adaptation of the search-tree algorithm of Suderman [182]. In both algorithms, each time the search-tree branches, the feedback edge set shrinks in each branch, allowing us to bound the depth of the search-tree in f. While the second algorithm has a better asymptotic running time, the first algorithm is much simpler and easier to implement.
In the algorithms, we use the notion of *branching rules* (see Section 1.2.4). Furthermore, we denote the *non-bridge degree* of a vertex v in G, that is, the number non-bridges incident to v in G, by $\deg_G^{\bigcirc}(v)$.

2.4.1 Straightforward Branching Algorithm

The algorithm runs in three phases. First, we apply a search-tree enumerating partial solutions by branching on a certain type of 2-claw. Second, we branch on small cycles in the remaining graph. In the third phase, we branch on the tokens (see Section 2.3.2, page 38) that remain in the graph. The input graph G considered in each phase is assumed not to be subject to any previous phase, that is, G does not contain a structure that is branched on in a previous phase. Furthermore, the input graph of each phase is assumed to be reduced with respect to the data reduction rules presented in the previous section. We show that the

total number of edge deletions done by branching in all three phases does not exceed the feedback edge set number of the input graph G. In the following, we present the three phases of our algorithm.

First Phase. In this paragraph, we describe the phase of the algorithm that branches on short cycles and 2-claws that do not contain bridges (that is, edges whose deletion disconnects G). Note that the bridges of G can be marked in linear time. Now, a vertex v with $\deg^{\bigcirc}_G(v) \geq 3$ is contained in a cycle of length at most four or a 2-claw that does not contain bridges (see Lemma 2 in [69]). We use this fact in the following branching rule.

Branching Rule 2.1. *Let $v \in V(G)$ with $\deg^{\bigcirc}_G(v) \geq 3$.*

1. *If v is contained in a cycle of length at most four, then, for each edge e on this cycle, create the partial solution $\{e\}$.*
2. *If v is contained in a 2-claw that does not contain bridges, then, for each edge e in this 2-claw, create the partial solution $\{e\}$.*

Clearly, Branching Rule 2.1 is correct and its branching number is at most 6. Since no partial solution contains a bridge, the minimum feedback edge set number decreases by one in each branch of the search tree. Note that, if G is reduced with respect to Branching Rule 2.1, then all cycles in G are vertex disjoint.

Observation 2.4. *If G is not subject to Phase 1, then all cycles in G are vertex-disjoint.*

Second Phase. With the first branching done, the next step is to eliminate all small cycles in the remaining graph. In this context, "small" means at most four vertices. We do this to assure that there is a remaining cycle that contains a token. Since all cycles in G are vertex-disjoint, we can find a cycle C of length at most four in G in $O(|G|)$ time. Then, we can branch on the at most four possibilities to destroy C by edge deletion.

Branching Rule 2.2. *Let G be reduced with respect to Branching Rule 2.1 and let C be a cycle of length at most four in G. Then, for each edge e in C, create the partial solution $\{e\}$.*

Since C is a cycle, no partial solution contains a bridge. Thus, the minimum feedback edge set number decreases by one in each branch. Since the target graph is a forest, we also know that one of the edges of C must be deleted, implying correctness of Branching Rule 2.2. As stated above, Branching Rule 2.2 can be applied in linear time.

Since, in this phase, G is reduced with respect to to Branching Rule 2.1, it is basically a tree of cycles and singletons. If G contains a cycle, then arbitrarily rooting this tree enables us to consider a cycle of G that has maximum distance from this root. Clearly, this cycle contains a vertex u such that $P_{u,u}$ is a degree-2 path (see Section 2.3.2, page 33 for the definition of degree-2 paths). If G is also not subject to Phase 2, then $P_{u,u}$ contains at least five inner path vertices. However, since G is reduced with respect to Path Reduction Rules 2.1 and 2.2, we know that $P_{u,u}$ cannot contain a vertex whose pendant tree is a Y-graph. Furthermore, since G is reduced with respect to Path Reduction Rule 2.3, we also know that $\mathcal{K}(P_{u,u}) \neq \emptyset$ (see Section 2.3.2, page 38 for the definition of tokens). Finally, if $\mathcal{K}(P_{u,u})$ contained only two tokens and both contained $\diamond^{\text{up}}$ or $\diamond^{\text{low}}$, then both tokens together can only contain six edges of $P_{u,u}$. However, since $|E(P_{u,u})| \geq 6$, the two tokens do not overlap, implying that there is a token-separator in $P_{u,u}$, which contradicts G being reduced with respect to Path Reduction Rule 2.4. Hence, there is a token in $\mathcal{K}(P_{u,u})$ that does not contain $\diamond^{\text{up}}$ or $\diamond^{\text{low}}$.

Observation 2.5. *Let G be reduced with respect to Branching Rules 2.1 and 2.2 and G contains a cycle. Then, there is a cycle of G that contains a vertex generating a non-auxiliary token.*

Third Phase. In the final part of the algorithm, we branch on the tokens in the remaining cycles. Recall that, if the input graph is acyclic, then Lemma 2.2 implies that its size is constant. Hence, we assume G to contain a cycle. Then, by Observation 2.5, some vertex of G generates a token K that we can use for branching.

Branching Rule 2.3. *Let G be reduced with respect to Branching Rules 2.1 and 2.2 and G contains a cycle. Let v be a vertex generating a non-auxiliary token K. Then, for each edge $e \in K$, create the partial solution $\{e\}$.*

Since the tokens of each vertex can be computed in O(1) time, we can apply Branching Rule 2.3 in linear time. By definition, $|K| \leq 4$ and, thus, Branching Rule 2.3 has branching number at most four. Since K is non-auxiliary, all of the edges of K are edges of $P_{u,u}$ and, hence, they are edges of a cycle, implying correctness of Branching Rule 2.3.
If we cannot find any further cycles in G, then G is a forest. However, since G is reduced with respect to the data reduction rules, we know that in this case, G has constant size (by Lemma 2.2) and, thus, can be solved in constant time.

Running Time. The algorithm presented in the previous paragraphs consists of three branching rules. Branching Rule 2.1 has the highest branching number,

hence, in the worst case, only Branching Rule 2.1 applies. Hence, the size of the search tree is $O(6^f)$. Since each branching rule can be executed once in linear time and the kernelization, which is to be applied in each branching step, runs in $O((f+1)\cdot|G|)$ time, we conclude that the algorithm runs in $O(6^f\cdot(f+1)\cdot|G|)$ time.

Theorem 2.2. Two-Layer Planarization *can be solved in* $O(6^f \cdot (f+1) \cdot |G|)$ *time.*

By initially applying the kernelization (see Section 2.3), we can bound the total running time by $O(6^f \cdot f^2 + (f+1) \cdot |G|)$.

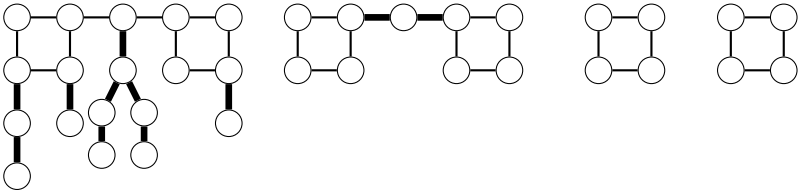
Corollary 2.2. Two-Layer Planarization *can be solved in* $O(6^f \cdot f^2 + (f+1) \cdot |G|)$ *time.*

2.4.2 Improved Branching Algorithm

In this section, we present a non-trivial adaptation of the algorithm of Suderman [182], which runs in $O(3.8^k \cdot |G|)$ time.[16] Suderman [182] defined five branching structures and developed a branching rule for each of them (see Figure 2.11, page 52). However, the branching vectors of these rules are with respect to the solution size k and not with respect to the feedback edge set number f. We use the same structures for our branching, but, in order to maintain the branching vectors, we ensure that the feedback edge set number decreases by the size of the corresponding partial solution in each branch. To this end, we find specific locations of branching structures in the input graph that allow this kind of branching. However, the price we pay for this advantage is that it becomes possible that none of the branching rules apply to the input graph. To deal with such graphs, we augment the process with a reduction rule that applies to graphs reduced with respect to the branching rules, thereby solving the given instance.

In the following, we first describe our modified versions of Suderman's branching rules and prove that they have the desired branching vectors. Applying these branching rules exhaustively leaves graphs with very specific properties. We exploit these properties in a data reduction rule that shrinks all graphs to which no other rules (branching or data reduction) apply. By repeated application, this rule either produces a graph that allows application of other rules or reduces the graph to constant size. Throughout the section, G denotes the input graph, G^* denotes its cyclic core (see Definition 2.1), and $G^{\bigcirc}$ denotes the result of removing all bridges from G.

[16] Note that Suderman [182] provided a refined algorithm running in $O(3.562^k \cdot |G|)$ time that we could not adapt for our parameterization.



(a) Graph G; bold edges are A-bridges

(b) Graph G^*; bold edges are B-bridges

(c) Graph $G^{\bigcirc}$

Figure 2.10: Illustration of the graph G^* that results from G by repeatedly removing degree-one vertices, and the graph $G^{\bigcirc}$ that results from G by removing all bridges. Bold edges in Figure 2.10a are A-bridges, bold edges in Figure 2.10b are B-bridges.

2.4.2.1 Adapting Previous Branching Rules

The first branching rule of Suderman [182] branches on cycles of length at most three and, thus, already decreases the feedback edge set number in each branch.

Branching Rule 2.4. *Let u, v, w be vertices in G forming a cycle. Then, create the partial solutions $\{\{u, v\}\}$,$\{\{v, w\}\}$, and $\{\{w, u\}\}$.*

In the following, we prepare the presentation of our core branching rules. To this end, we develop a strategy to apply the branching rules of Suderman [182] while avoiding to branch on bridges of G. This is necessary since, by definition, deleting an edge e decreases the feedback edge set number by one if and only if e is not a bridge. Here, two obstacles arise when trying to maintain the branching vectors of the branching rules of Suderman [182]: First, there may be a bridge of G in a partial solution. Second, a partial solution may contain two edges but deleting one of them makes the other a bridge, implying that the feedback edge set number decreases by one instead of two.
By carefully selecting a branching structure, we can avoid bridges of G. To this end, we differentiate between two kinds of bridges in G. The first kind are edges in $G \doteq E(G^*)$, that is, edges of pendant trees. We call them *type A bridges* or simply *A-bridges*. The second kind are the bridges in G^*. We will call them *type B bridges* (*B-bridges*). Figure 2.10 gives an intuition on the definition of both kinds of bridges.
Lemma 2.6 states that there are optimal solutions that do not delete A-bridges unless they belong to Y-graphs, implying that not all A-bridges pose a problem to the branching rules. In fact, partial solutions that contain such an edge can be

ignored. As it turns out, some of these remaining A-bridges are not as problematic as others. Consider a Y-graph T^v with $v \in V(G^*)$. If v is incident to exactly two non-bridges and no B-bridge,[17] then we can assume an optimal solution to delete either the A-bridge or both non-bridges. This allows us to discard branches that delete both the A-bridge and any of the two non-bridges.

Lemma 2.11. *Let Y be a pendant Y-graph with connection point v and center w such that v is incident to exactly two non-bridges e_1 and e_2 and no B-bridge in G. Then, there is an optimal solution S for $G \dot{-} \{e_1, e_2\}$ such that either $S \cup \{\{v, w\}\}$ or $S \cup \{e_1, e_2\}$ is an optimal solution for G.*

Proof. Let $G' = G \dot{-} \{e_1, e_2\}$ and let $H := G[N_G[v] \cup V(Y)]$. Note that H contains a 2-claw centered at w. Consider an optimal solution S^* for G and let $S_H := S^* \cap E(H)$. Clearly, $S' := S^* \setminus S_H$ is a solution for G'. Since H contains a 2-claw, we know that $|S_H| > 0$. If $|S_H| = 1$, then S_H does not contain e_1 or e_2 since in both cases $H - S_H$ contains a 2-claw. Clearly, deleting $\{v, w\}$ is at least as good as deleting any other edge of Y and, thus, we can assume $S_H = \{v, w\}$, and the claim of the lemma follows. If $|S_H| = 2$ then, since $G' = G \dot{-} \{e_1, e_2\}$, we know that $S' \cup \{e_1, e_2\}$ is a solution for G and we know that $|S^*| = |S'| + 2$. Thus, by optimality of S^*, we conclude that $S' \cup \{e_1, e_2\}$ is optimal for G. □

In the following, we call an A-bridge $\{v, w\}$ *relevant* if there is a Y-graph with connection point v and center w that does not satisfy the conditions of Lemma 2.11, that is, v is incident to B-bridges or more than two non-bridges.
Compared to the branching rules of Suderman [182], we have to select branching structures more carefully. In the following, we describe how to find branching structures in G such that applying a created partial solution decreases $f(G)$ by the size of the partial solution. To this end, let $v \in V(G^{\bigcirc})$. We define edges incident to v that can be included in partial solutions.

Definition 2.3. *Let $v \in V(G^{\bigcirc})$. We call v* branchable *if there are three non-leaves $v_1, v_2, v_3 \in N_G(v)$ such that*
(1) For each $1 \leq i \leq 3$, $\{v, v_i\}$ is not a relevant A-bridge.
(2) For each $1 \leq i \leq 3$, there is no B-bridge incident to v_i.
We call v_1, v_2, and v_3 branching partners *of v.*

By finding a vertex with degree at least three in the graph that remains after deleting all relevant A-bridges and all B-bridges from G, a branchable vertex and its branching partners can be found in linear time. If all three branching partners of v are in $G^{\bigcirc}$, then we require some additional properties of branching partners

[17] Note that, by Lemma 2.2, there is no other A-bridge incident to v in G.

to prove the desired branching vectors. Fortunately, it is not hard to establish these properties for given branching partners of v.

Lemma 2.12. *Let $v \in V(G^{\bigcirc})$ be branchable and v has three branching partners in $G^{\bigcirc}$. Then, there are three branching partners $v_1, v_2, v_3 \in V(G^{\bigcirc})$ of v such that*
(1) $\deg_G(v_1) = 2 \Rightarrow \deg_G(v_2) = 2$ *and* $\deg_G(v_2) = 2 \Rightarrow \deg_G(v_3) = 2$.
(2) if $\deg_G(v_1) = 2$, *then deleting* $\{v, v_1\}$ *does not make* $\{v, v_2\}$ *a bridge, and*
(3) if $\deg_G(v_1) > \deg_G(v_2) = 2$, *then deleting* $\{v, v_2\}$ *does not make* $\{v, v_3\}$ *a bridge.*

Proof sketch. Given branching partners v_1, v_2, and v_3 for v, we show how to reorder and reassign them to abide by Lemma 2.12.
First, to establish (1), we can simply sort v_1, v_2, and v_3 by their degree in G. Second, to establish (2), we can just swap v_2 and v_3 if necessary. Finally, to establish (3), note that if deleting $\{v, v_2\}$ makes $\{v, v_3\}$ a bridge, then, deleting both $\{v, v_2\}$ and $\{v, v_3\}$ does not make $\{v, v_1\}$ a bridge. Since $v_1 \in V(G^{\bigcirc})$, there is some $v_4 \in N_{G^{\bigcirc}}(v)$ such that $\{v, v_4\}$ is not a bridge in G. This allows us to replace v_2 by v_4, while $\deg_G(v_1) > 2$ ensures that both (1) and (2) remain intact. □

In the following, we assume that v is branchable and its branching partners v_1, v_2, and v_3 fulfill all properties of Definition 2.3 and Lemma 2.12. To complete the branching structure, it remains to select appropriate edges incident to v_1, v_2, and v_3. To this end, we define sets E_1, E_2, and E_3 of edges incident to v_1, v_2, and v_3, respectively.

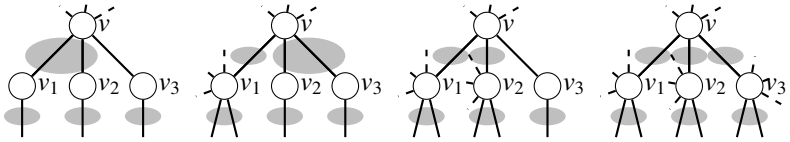
Definition 2.4. *For $1 \leq i \leq 3$, let $e_i = \{v, v_i\}$ and let E_i denote a set of at most two edges of G such that*
(i) all edges in E_i are incident to v_i but not to v,
(ii) if $|E_i| = 1$, then $\deg_G(v_i) = 2$,
(iii) if $\deg_{G^{\bigcirc}}(v_i) \geq 3$, then there is an edge $\{v_i, u\} \in E_i$ such that u is connected to v via a path in G that avoids v_i, and
(iv) if there is a bridge of G in E_i, then $\deg_{G^{\bigcirc}}(v_i) = 2$.

Algorithm 2.1 computes E_i for given v and v_i in linear time. The bottleneck is finding a suitable non-bridge that respects Definition 2.4(iii) in line 6 that can be implemented by a graph traversal. We can now state the modified versions of the branching rules of Suderman [182].

Branching Rule 2.5 (based on "CLAW0" [182] shown in Figure 2.11a)**.** *Let $|E_1| = 1$. Then, create the partial solutions E_1, E_2, E_3, and $\{e_1, e_2\}$. Discard all partial solutions containing non-relevant A-bridges.*

Algorithm 2.1: Given v and v_i, compute E_i in $O(|G|)$ time.

```
1 N_i ← non-bridges incident to v_i except for {v, v_i};
2 if deg_{G^○}(v_i) = 2 then
3 |   E_i ← N_i;
4 |   if there is a bridge a incident to v_i then add a to E_i;
5 else if deg_{G^○}(v_i) ≥ 3 then
6 |   E_i ← a non-bridge in N_i that respects Definition 2.4(iii);
7 |   add a non-bridge in N_i \ E_i to E_i;
```



(a) CLAW0 (b) CLAW1 (c) CLAW2 (d) CLAW3

Figure 2.11: Schematic view on four of the first five branching rules of Sud-erman [182] (his rule called "3CYC" is omitted since it is equal to Branching Rule 2.4). Gray ellipses indicate the edges deleted in each branch (each ellipse corresponds to a different branch). The branching vectors are (1, 1, 1, 2), (1, 1, 1, 2, 2), (1, 1, 1, 2, 2), and (1, 1, 1, 2, 2, 2).

Branching Rule 2.6 (based on "CLAW1" [182] shown in Figure 2.11b)**.** *Let* $|E_1| > |E_2| = 1$. *Then, create the partial solutions* E_2, E_3, $\{e_1\}$, *and* $\{e_2, e_3\}$. *If* E_1 *does not contain an A-bridge, then additionally create the partial solution* E_1. *Discard all partial solutions containing non-relevant A-bridges.*

Branching Rule 2.7 (based on "CLAW2" [182] shown in Figure 2.11c)**.** *Let* $|E_2| > |E_3| = 1$. *Then, create the partial solutions* E_3, $\{e_1\}$, *and* $\{e_2\}$. *For each* $1 \leq i \leq 2$, *if* E_i *does not contain an A-bridge, then additionally create the partial solution* E_i. *Discard all partial solutions containing non-relevant A-bridges.*

Branching Rule 2.8 (based on "CLAW3" [182] shown in Figure 2.11d)**.** *Let* $|E_3| > 1$. *Then, create the partial solutions* $\{e_1\}$, $\{e_2\}$, *and* $\{e_3\}$. *For each* $1 \leq i \leq 3$, *if* E_i *does not contain an A-bridge, then additionally create the partial solution* E_i. *Discard all partial solutions containing non-relevant A-bridges.*

The correctness proof of Branching Rules 2.5–2.8 is based on the correctness of the original rules [182].

Lemma 2.13. *Branching rules 2.5–2.8 are correct, that is, for each of these branching rules, one of the created partial solutions can be extended to an optimal solution for G.*

Proof. For the sake of contradiction, assume that none of the created partial solutions can be extended to an optimal solution for G. Hence, by correctness of the branching rules of Suderman [182], there is some $1 \leq i \leq 3$ such that E_i is not created and E_i can be extended to an optimal solution S^* for G. Since E_i is not created, E_i contains an A-bridge b which, by Lemma 2.6, can be assumed to be relevant. Hence, Definition 2.4(iv) implies $\deg_{G^\bigcirc}(v_i) = 2$ and, thus, Lemma 2.11 is applicable. Then, however, we can replace b with e_i in S^*, implying that the partial solution $\{e_i\}$, which is created in all branching rules in question, can be extended to an optimal solution for G. □

We can show that the worst-case branching vector $(1, 1, 1, 2, 2, 2)$ (see Figure 2.11) is matched by our branching rules. First, we consider the case that a branching partner of v is not in $G^\bigcirc$ and show a worst-case branching vector of $(1, 1, 2, 2)$ in this case. Then, we prove that in case all branching partners are in $G^\bigcirc$, the feedback edge set number decreases by the number of edges deleted. Thus, our branching vectors equal those shown by Suderman [182].

Lemma 2.14. *If one of the branching partners of v is not in $G^\bigcirc$, then the branching number of Branching Rules 2.5–2.8 is at most* 2.733.

Proof. Let $V' := \{v_1, v_2, v_3\} \setminus V(G^\bigcirc)$. By Definition 2.3, no $\{v, v_i\}$ is a relevant A-bridge or a B-bridge Hence, by the statement of the branching rules, all partial solutions containing edges incident to vertices in V' are discarded. Thus, if $|V'| \geq 2$, then the worst-case branching vector of the branching rules is $(1, 1)$, corresponding to a branching number of two. Hence, assume $|V'| = 1$. Consider $|E_i|$ for all $v_i \notin V'$ and note that E_i does not contain a bridge of G because, by Definition 2.3, branching partners of v are not incident to B-bridges and the branching rules do not create E_i if it contains A-bridges. By Definition 2.4(iii), deleting E_i decreases the feedback edge set number by $|E_i|$.
In the following, let $V' = \{v_x\}$ and recall that E_x and any partial solution containing e_x contain non-relevant A-bridges and are, therefore, discarded. Consider the branching rules separately:
Case 1: Branching Rule 2.5 applies to v. Then, by symmetry, we may relabel v_1, v_2, v_3 such that $v_x = v_2$. This implies a branching vector of $(1, 1)$ corresponding to a branching number of two.
Case 2: Branching Rule 2.6 applies to v. Then, either $v_x = v_1$, implying a branching vector of $(1, 1, 2)$ or $v_x \in \{v_2, v_3\}$, implying a branching vector of $(1, 1, 2)$. In both cases, the branching number is 2.415.

Case 3: Branching Rule 2.7 applies to v. Then, either $v_x \in \{v_1, v_2\}$, implying a branching vector of $(1, 1, 2)$ or $v_x = v_3$, implying a branching vector of $(1, 1, 2, 2)$. In both cases, the branching number does not exceed 2.733.
Case 4: Branching Rule 2.7 applies to v. Then, the branching vector is $(1, 1, 2, 2)$, corresponding to a branching number of 2.733. □

Lemma 2.15. *Let $v_1, v_2, v_3 \in V(G^{\bigcirc})$ and let S be a partial solution created by one of Branching Rules 2.5–2.8. Then, removing the edges in S decreases the feedback edge set number of G by $|S|$.*

Proof. First, consider the case that $|S| = 1$, that is, $S = \{e\}$. If e is incident to v, then, since $v_1, v_2, v_3 \in V(G^{\bigcirc})$, e is not a bridge. If e is not incident to v, then, $e \in E_i$ for some $1 \leq i \leq 3$. Since partial solutions E_i are only created if they do not contain A-bridges and, by Definition 2.3, e is not a B-bridge, e is not a bridge. In both cases, removing e decreases the feedback edge set number by one.
In the following, we assume $|S| = 2$, that is $S = \{e_1, e_2\}$. First, let S consist of edges incident to v. Since $v_1, v_2, v_3 \in V(G^{\bigcirc})$, neither e_1 nor e_2 are bridges. Towards a contradiction, assume that deleting an edge of S makes the other a bridge. Note that either $S = \{e_1, e_2\}$ in Branching Rule 2.5 or $S = \{e_2, e_3\}$ in Branching Rule 2.6. The first case contradicts Lemma 2.12(2), the second case contradicts Lemma 2.12(3).
In the following, we assume that the edges of S are not incident to v, that is, $S = E_i$ for some $1 \leq i \leq 3$. Then, by Definition 2.3, E_i does not contain B-bridges and by the statements of the branching rules, E_i does not contain A-bridges. Thus, E_i consists of two non-bridges. Hence, $\deg_{G^{\bigcirc}}(v_i) \geq 3$ and, by Definition 2.4(iii), there is an edge $e = \{v_i, u\}$ in E_i such that there is a path from u to v that avoids v_i. Let $E_i = \{e, e'\}$. Clearly, deleting e' does not make e a bridge, since, otherwise, all paths from u to v would contain e' and, therefore, also v_i, contradicting the choice of e. Hence, deleting E_i decreases the feedback edge set number by two. □

2.4.2.2 Reducing the Remaining Graph

In the following, we consider graphs that are reduced with respect to our branching rules, that is, graphs to which none of the presented branching rules are applicable. This is mostly due to branching partners not being incident to B-bridges (see Definition 2.3). Thus, we present a way to deal with B-bridges. To this end, consider the tree that results from G by contracting each connected component of $G^{\bigcirc}$ to a single vertex. We call this tree the "component tree" T^C of G. By considering a leaf in T^C, we can limit the possibilities for branching structures to contain B-bridges. Note that T^C can be computed in linear time.

In the following, we consider graphs G that are reduced with respect to the presented data reduction and branching rules. Consider a leaf L in the component tree T^C of G and let u denote the only vertex in L that is incident to a B-bridge in G. We can observe the following properties of G.

Observation 2.6. *Let G be reduced with respect to all presented branching and reduction rules. Let L be a leaf in T^C and let $u \in L$ with $N_G(u) \not\subseteq L$. Then,*
(a) $G[L]$ does not contain B-bridges;
(b) $|N_{G^\bigcirc}(u)| = 2$ because, otherwise, we could apply a branching rule to u;
(c) each vertex $x \in L \setminus N_{G^\bigcirc}[u]$ has at most two non-leaf neighbors in G since, otherwise, we could apply a branching rule to x;
(d) the two vertices in $N_{G^\bigcirc}(u)$ have the same degree in $G^\bigcirc$, since by (c) and (b) all degree-2 paths starting in one of them must end in the other.

Observation 2.6 fixes the structure of $G[L]$ which we can exploit with the following data reduction rule. In the following, we call a vertex *dirty* if its pendant tree is not a leaf, a singleton or a Y-graph.

Reduction Rule 2.1. *Let G be reduced with respect to all presented data reduction and branching rules and let L be a leaf in the component tree of G such that the vertex u is the only vertex in L that is incident to a B-bridge in G. Let $\{v, w\} = N_{G^\bigcirc}(u)$ such that w being dirty implies v being dirty. If $\deg_{G^\bigcirc}(v) = 2$ and w is dirty, then delete an edge of $G^\bigcirc[L]$ with maximum distance to u. Otherwise, delete $\{u, v\}$. In both cases, decrement k by one.*

Lemma 2.16. *Reduction Rule 2.1 is correct and can be applied in linear time.*

Proof. First, consider the case that $\deg_{G^\bigcirc}(v) = 2$ and w is dirty. Then, by choice of v, also v is dirty. Hence, by Observation 2.6(c), $G^\bigcirc[L]$ is a degree-2 path from u to u. There are no dirty vertices in $L \setminus \{u, v, w\}$, since otherwise, we could apply a branching rule to this vertex. Hence, by reducedness with respect to Path Reduction Rule 2.4, the 2-claws centered in v and w in G overlap in an edge with maximum distance to u. Deleting an edge from $G[L]$ that does not have maximum distance to u does not destroy both 2-claws centered at v and w. Clearly, if any optimal solution contains two edges of $G[L]$, then replacing these two edges with an edge with maximum distance to u and the B-bridge incident to u yields an optimal solution for G.
In the following, we assume that $\deg G^\bigcirc(v) > 2$ or w is not dirty. We prove that there is an optimal solution S for G that contains $\{u, v\}$. We will use the following arguments in the proof.
(1) If there is an optimal solution for G containing $\{u, v\}$, then we are done. Hence, we assume that no optimal solution contains $\{u, v\}$. Then, however,

for all solutions S, all 2-claws in $G \dot{-} ((S \setminus E(G[L])) \cup \{\{u, v\}\})$ have their center in L, that is, "shuffling" edge deletions in $G[L]$ does not create a 2-claw whose center is not in L, as long as $\{u, v\}$ is deleted.

(2) Let S^* be an optimal solution for G and assume that, for some pendant Y-graph Y of a vertex z in $G^* - v$, S^* contains $\{z, c(Y)\}$. Then, there is an optimal solution for G containing $\{u, v\}$ if and only if there is an optimal solution for $G \dot{-} \{z, c(Y)\}$ containing $\{u, v\}$. Thus, in the following, we assume that no optimal solution contains an edge of a pendant Y-graph except for the pendant Y-graph of v.

(3) If there is a vertex z with a pendant Y-graph in L, then, by (2), all optimal solutions contain $\deg_G(z) - 1$ edges incident to z. However, deleting $\deg_G(z) - 2$ of these makes the last one a bridge. Hence, the feedback edge set number decreases by only $\deg_G(z) - 2$. Thus, the feedback edge set number of $G[L]$ plus the number of pendant Y-graphs in $G[L]$ is a lower bound for the size of a solution for $G[L]$.

(4) The vertex u is not dirty, since otherwise, by Observation 2.6(b) the non-leaf degree of u in $G[L]$ is at least three, implying that we could apply a branching rule to u.

Let f_L denote the feedback edge set number in $G[L]$ and let $Z \subseteq E$ denote the set of all relevant A-bridges in $G[L]$. Then, by Observation 2.6(c and d), $f_L = \deg_{G^*}(v) - 1$. By (3), an optimal solution for $G[L]$ contains at least $|Z| + \deg_{G^*}(v) - 1$ edges. We construct a solution S for $G[L]$ that contains $\{u, v\}$ and matches this lower bound and show that S can be extended to an optimal solution for G.

If $f_L > 2$, then, by Observation 2.6(c), $\deg_{G^\circ}(v) > 3$, implying that we could apply a branching rule to v in G. Hence, in the following, we assume that $1 \leq f_L \leq 2$. If $f_L = 1$, let $S := Z \cup \{\{u, v\}\}$ and note that, by definition, w is not dirty. If $f_L = 2$, let $S := Z \cup \{\{u, v\}, \{w, z\}\}$ for some $z \in N_{G^\circ}(w) - u$ and note that w is not dirty, since otherwise, we could apply a branching rule to w in $G \dot{-} \{u, w\}$. Since in both cases $\deg_{G^\circ \dot{-} S}(w) < 3$ and $Z \subset S$, we conclude that w has at most two non-leaf neighbors in $G \dot{-} S$.

We show that S is a solution for $G[N_G[L]]$. If this is not the case, then there is a 2-claw centered at some $x \in L$ in $G[N[L]] \dot{-} S$. Clearly, $x \in N_{G[L]}[u]$, since, otherwise, we could have applied a reduction rule to x in G. By (4), $x \neq u$. Since $Z \subset S$, we conclude $x \in \{v, w\}$. Since w has at most two non-leaf neighbors in $G \dot{-} S$, we conclude $x \neq w$ and, hence, $x = v$. However, since $\{u, v\}$ is not in $G[L] \dot{-} S$, we could apply a reduction rule to v in G, contradicting reducedness of G. Since $|S| = |Z| + f_L$, by (3), S is an *optimal* solution for $G[N[L]]$. Let S^* denote an optimal solution for G and let b denote the B-bridge incident to u. If $|S^* \cap E(G[N[L]])| > |S|$, then $S \cup \{b\}$ can be extended to an optimal solution for G. Otherwise, $G \dot{-} (S \cup (S^* \setminus E(G[N[L]]))$ contains a 2-claw, which, by (1)

and S being an optimal solution for $G[N[L]]$, is centered at u and contains b. However, by (4), the 2-claw contains $\{u, w\}$ and $\{u, v\}$, contradicting $\{u, v\} \in S$. To prove that Reduction Rule 2.1 can be applied in linear time, recall that all leaves of the component tree of G can be found in linear time and then solved individually. Clearly, for each leaf L, we can check the conditions in $O(1)$ time and apply the deletion in $O(|L|)$ time, implying linear time overall. □

Since Reduction Rule 2.1 can be applied whenever none of the other reduction or branching rules can be applied, applying all presented rules exhaustively solves the input instance. The worst-case branching vector corresponds to Branching Rule 2.8 and is $(1, 1, 1, 2, 2, 2)$. This implies a search tree with 3.8^f nodes. As noted earlier, computing the sets E_1, E_2, and E_3 takes linear time per vertex. By running the kernel presented in Section 2.3 in the beginning, we can limit the time spent in each search-tree node by $O(f^2)$. This, however, implies an additional time of $O(f \cdot |G|)$.

Theorem 2.3. Two-Layer Planarization *can be solved in* $O(3.8^f \cdot f^2 + f \cdot |G|)$ *time, where* f *denotes the feedback edge set number of the input graph.*

2.5 Heuristic Speedups and Experimental Results

In this section, we describe the setting of our experiments. In particular, we describe heuristic improvements of the algorithm that we implemented, explain how our test-instances were generated, and give details of the machine, operating system, programming language, and compiler settings we used in our tests.

2.5.1 Heuristic Speedups

In the following, we describe heuristic tricks that we used in our implementation to speed up the computation of the size of an optimal solution.

Extending the Sets E_i. Observe that the correctness proofs of Suderman [182] for the branching rules we employ are not limited to $|E_i| \leq 2$. They work just as well if E_i contains all edges incident to v_i except $\{v, v_i\}$. Hence, we extended the sets E_i accordingly. Note that, since the new sets E_i' are supersets of the sets E_i, the branching vectors of Branching Rules 2.5-2.8 improve.

Analyzing and Sorting Branching Vectors. In each search-tree node, we are challenged with finding a "good" branching structure to continue our search for an optimal solution. A branching structure is *good* if the smallest branching

number of any applicable branching rule is small. Our strategy is to find all reasonable branching structures, sort them by their branching number, and branch using the best possible branching rule. Here, "reasonable" means that, we find at most one application of Branching Rule 2.4 since each such application has branching number three. We use C-code of Chuang-Chieh Lin [51] to compute the branching number of a branching vector. As soon as an application of a branching rule with branching number one is found, we cancel the search for other branching structures and use this one.

Additional Reduction. Lemma 2.6 tells us that, in graphs reduced with respect to the tree reduction rules, we do not need to consider partial solutions containing non-relevant A-bridges. Thus, if we find a branching structure such that all but one of its edges are non-relevant A-bridges, then the branching rule degenerates to a reduction rule.

Reduction Rule 2.2. *Let $\{u, v\}$ be an edge of the cyclic core G^* of G such that the pendant of u contains two P_2s and the pendant of v is not a singleton. Then, delete $\{u, v\}$ and decrease k by one.*

The correctness of Reduction Rule 2.2 follows from the discussion above. In fact, with the heuristic improvement of the previous paragraph, our algorithm implicitly applies Reduction Rule 2.2. However, explicit application of Reduction Rule 2.2 saves some overhead since we avoid calling the subroutines for the branching rules.

Computing each Component Separately. In each search-tree node, we use a linear-time algorithm of Tarjan [186] to find and mark all bridges in the current graph G. This algorithm is also capable of detecting whether G is disconnected. If G contains multiple connected components, then an optimal solution is split among them, allowing us to return the sum of the sizes of optimal solutions for each component. This way, the number of leaves of the search-subtree rooted at the current search-tree node is the sum of the leaves in the search-trees of the connected components of G, instead of the product.

Branch & Bound with f as Lower Bound. We keep track of an optimal solution found by our algorithm so far. If any branch cannot contain a better solution, then we cancel the branching and return failure back to the parent of the search-tree. Lower-bound techniques are used to determine whether a better solution is possible in this branch. We tested different algorithms and found that the best overall performance was delivered by simply using the feedback edge set

number f as a lower bound. On the one hand, this is not a good bound, since f can be far from the solution size k. On the other hand, f can be computed very quickly (indeed, the previously mentioned scan for bridges we perform in each search-tree node already provides this).

Permanent Edges. If a partial solution contains a single edge e then, after searching the search-subtree corresponding to the deletion of this edge, we can mark it "permanent". This means that all optimal solutions for the current graph G that contain e have been considered, so e can be excluded from further branching, thereby improving the branching vectors.

2.5.2 Experiments

In this section, we discuss an implementation of our algorithm presented in Section 2.4.2. The program solves the optimization variant of Two-Layer Planarization, which asks for the minimum number of edge deletions to make the input graph biplanar. The program contains implementations of all branching and data reduction rules presented in Section 2.3 and Section 2.4.2. It comprises about 3100 lines of code written in the C++ programming language.
For comparability of results, we followed the example of Suderman and Whitesides [183] and included the size of the search tree in the results, since this value is a measure of speed that depends only on the algorithm, not the hardware.

Machines and Settings. The tests were run on an Intel(R) Xeon(R) E5-1620 CPU at 3.6GHz without taking advantage of the multiprocessor capabilities. The systems were running Debian Linux with the following software versions.

GNU/Linux	3.2.0
GNU libc	2.13
gcc	4.7.1

The program was compiled with `CFLAGS=-march=native -msahf -O3`.

Instance Generation. We studied two test-case scenarios. First, we reproduced the generated instances used by Mutzel [156], Suderman and Whitesides [183], and Suderman [182] (where detailed descriptions on reproducing the instances can be found). This test set comprises 1700 "dense" bipartite graphs $(V_1 \uplus V_2, E)$ with $|V_1| = |V_2| = 20$ and $|E|$ between 20 and 100 and 900 "sparse" bipartite graphs with $|V_1| = |V_2|$ between 20 and 100 and $|E| = 2|V_1|$. As shown in Table 2.1, feedback edge set numbers are at most 60 for dense graphs and 32 for sparse

graphs. Note that solution sizes are actually not much larger than the feedback edge set numbers for these instances.

The second set of instances comprises 1500 large, sparse graphs that were generated by for each $n \in \{100i : 1 \leq i \leq 10\}$ and each $p \in \{3\%, 6\%, 9\%, 12\%, 15\%\}$, constructing a tree on n vertices and adding $p \cdot n$ edges uniformly at random. If some insertion failed because the edge was already present, we repeated the insertion with new random values so that the graphs are guaranteed to contain $n-1+p \cdot n$ edges.

It turned out in the experiment that, for $p \leq 9\%$, all instances were solved in a matter of milliseconds with search-tree sizes below 10. Thus, these cases are not included in the table of results (Table 2.3). The way we created the instances implies that the feedback edge set number for each instance is fix. Solution sizes range between 20 and 225. Typically, the ratio of solution size to feedback edge set number is between 1.4 and 1.7.

Results. The results obtained by the branching algorithm of Suderman [182] and the ILP formulation of Jünger and Mutzel [130] are compared to the results of our implementation in Table 2.1. First, consider the set of "dense" graphs (first 17 rows). It is noteworthy that these graphs do not fit well in the picture we painted in the intoduction. More precisely, their feedback edge set number f differs from the solution size k by at most 2. This, however, cannot explain the dramatic differences in search-tree sizes between the two algorithms. In the following, we try to give an explanation of why our algorithm compares so poorly to Suderman's.

1. Looking at the search-tree sizes for the algorithm of Suderman, it quickly becomes apparent that they differ only marginally for all k between 14 and 61 leading to the conjecture that the search-tree sizes are influenced by some other, hidden factor.
2. It is easy to suspect that Suderman's algorithm may achieve smaller search-trees since it branches on bridges, thereby disconnecting the graph, allowing both parts to be processed individually. However, our algorithm avoids bridges completely, considering only those partial solutions that Suderman's algorithm would also have to consider later.
3. Since k and f are so close in these instances, constants in the kernelization-bound may have an influence on the effectiveness of the preprocessing. This becomes less likely when recalling that Suderman [182] showed a comparison of his branching algorithm to the previously known Dujmović et al. [69], which already incorporated the kernelization but could not solve any instances with at least 60 edges.

4. Suderman employs a very tight lower bound to cancel branches that cannot yield a better solution than what was already computed. Our lower bound, however, is simply the feedback edge set number of the current graph. If this was indeed the cause of the observed difference in search-tree sizes, then we could just replace our crude lower bound with Suderman's.
5. Finally, Suderman describes a sophisticated divide-and-conquer technique based on "p-components". While a mathematical analysis of this technique is open, Suderman described it as very effective and should be incorporable in our algorithm as well.

Note that our running times still rival those of Suderman's algorithm, which is to be expected considering that the results for the older algorithm were obtained in 2005 on a 1GHz Pentium III computer [182].

On the "sparse" instances, we expected our algorithm to perform better than on the set of dense graphs. In fact, we were able to solve a good portion of the larger instances that could not be solved in the past. On the one hand, this may again be due to our hardware advantage. On the other hand, we observe a larger divergence between the parameters f and k.

A closer inspection of the running times of our implementation revealed that Table 2.1 does not reflect the behavior of our algorithm very well. In particular, some small spikes in running time and search-tree size bring down the average performance of the algorithm. Therefore, we give a more detailed test in Table 2.2 and draw the (cumulative) distribution of running times on the set of dense graphs in Figure 2.12. Notice the striking difference between average and median running times and search-tree sizes that differ by a factor of up to $4,500$. Also note that, after about one second, more than 90% of all "dense" instances were solved (see Figure 2.12). The set of sparse graphs even contained instances of up to 60 vertices that were completely solved by the kernelization (search-tree size 1). Table 2.2 and Figure 2.12 raise hope that our algorithm will perform well on a wide range of inputs.

The performance of our algorithm on the second batch of instances is shown in Table 2.3. For graphs with up to 1000 vertices and $|E|/|V| \leq 1.12$, our algorithm always finishes within half a second. However, for $|E|/|V| = 1.15$, we could not even solve all instances containing 600 vertices. Again, the median running times paint a brighter picture. Half of all input instances with 1000 vertices and $|E|/|V| = 1.15$ were solved after about 2 minutes.

While we have to admit that the tested instances are very sparse, we also note that Suderman [182] performed tests on instances with $|E|/|V| = 0.6$. Furthermore, our algorithm is designed to run on sparse graphs. It seems unreasonable to try and run it on dense graphs. Instead, we consider it an interesting open question whether Two-Layer Planarization is fixed-parameter tractable and maybe even

admits a polynomial-size problem kernel with respect to a density measure that is small when the input graph is dense. Candidates may be the clique cover number, the distance to cluster graphs (see Section 1.2.4, page 21) or the cliquewidth [53] of the input. Note that this is of theoretical interest only, since, as mentioned before, practical instances can be expected to be sparse.
Last but not least, we want to get a glimpse of the efficiency of our implementation of the kernelization presented in Section 2.3. To this end, we plotted the time per search-tree node versus the size of the input graph in Figure 2.13. Although the time per search-tree node is also influenced by our elaborate method of selecting the best possible branching vector first, we estimate that the application of the reduction rules dominates the running time. Although no clear trend can be made out in Figure 2.13, times between 100μs and 200μs per search-tree node can be observed for all input sizes, suggesting a rather slowly growing function.

2.6 Conclusion

We presented a linear-size problem kernel and two search tree algorithms for Two-Layer Planarization parameterized by the "feedback edge set number", a structural parameter that is upper-bounded by the size of an optimal solution. The results represent a proof of concept for considering non-standard parameters, not only for parameterized algorithms but also for polynomial-time preprocessing. This is especially true since our asymptotic bounds with respect to f are equal (for kernelization) or very close (for the search-tree algorithm) to the best known bounds with respect to the standard parameter "solution size".
We implemented and tested the kernelization procedure in conjunction with our search-tree algorithm. We concluded that the previous algorithm by Suderman [182] is, on average, superior to our implementation on "dense" graphs. However, our algorithm is designed to perform well on *sparse graphs*, which we could demonstrate. Unfortunately, we could not pit our algorithm against Sudermans [182] on the treelike testgraphs. Our implementation could profit from Suderman's "p-component" technique which is worth looking into before running further comparisons. We interpreted the slow increase in running time per search-tree node (Figure 2.13 as an indicator for the efficiency of our kernelization implementation. It remains to back this up with a more thorough theoretical analysis.
Another interesting line of research is opened by replacing edge deletion as the allowed graph modification operation by the so-called "node duplication" operation[18], yields the Node Duplication based Crossing Elimination problem [43, 44, 45, 189], which has applications in the design of molecular quantum-dot cellular

[18] For a vertex v, duplicating v means to delete v and introduce u, u' with $N(u) \uplus N(u') = N(v)$.

automata [43] and in visualization of gene ontologies in bioinformatics [189]. It is interesting whether our kernelization and search-tree algorithm can be adapted to work for this problem as well.
In light of the linear-time algorithm for edge-weighted Two-Layer Planarization on trees presented by Shahrokhi et al. [174], it is interesting to investigate whether our kernelization approach also holds for the edge-weighted case.
Providing efficient fixed-parameter algorithms (in particular polynomial-size problem kernels) for parameters upper-bounded by the feedback edge set number is a natural next step to extend the range of solvable instances of Two-Layer Planarization. The feedback vertex set number would be a canonical candidate. Additionally, it may be promising to investigate the parameter $k' = (k - f)$ that represents an "above guarantee" parameter for the problem. Finally, extending our results to the multilayered problem versions [70] is an interesting endeavor.

				ILP	3.562^k search tree [182]			3.8^f search tree		
$\|V_i\|$	$\|E\|$	k	f	Time	Time	Steps	%	Time	Steps	%
20	20	1	0	0	0	2	100	0	1	100
20	25	2	1	0	0	2	100	0	1	100
20	30	3	2	0	0	5	100	0	2	100
20	35	6	4	1	0	14	100	0	4	100
20	40	8	6	6	0	76	100	0	26	100
20	45	11	9	26	0	85	100	0	157	100
20	50	14	13	100	4	4,694	100	0	897	100
20	55	18	17	81	1	946	100	0	2,417	100
20	60	23	22	56	5	6,232	100	1	18,596	100
20	65	27	27	54	3	3,645	97	8	305,501	99
20	70	32	31	26	7	8,263	99	14	489,962	98
20	75	37	36	22	2	2,249	100	4	147,080	99
20	80	41	41	12	2	2,060	99	1	27,630	99
20	85	46	46	20	5	5,366	100	2	82,563	99
20	90	51	51	8	6	6,503	99	2	89,623	99
20	95	56	55	4	8	8,276	99	3	126,372	97
20	100	61	60	4	4	5,243	98	1	37,733	96
20	40	7	6	6	0	95	100	0	24	100
30	60	11	10	49	0	356	100	0	231	100
40	80	16	13	150	3	3,002	100	0	1,546	100
50	100	19	16		14	11,876	99	1	21,754	99
60	120	24	19		64	48,240	96	2	37,182	99
70	140	28	23		129	91,339	88	6	112,015	99
80	160	31	26					22	339,282	90
90	180	35	29					44	661,619	91
100	200	38	32					74	1,097,335	81

Table 2.1: Results of the first batch of tests. The numbers in each row are averaged over the instances that were solved successfully. The column labeled "Steps" contains the numbers of search-tree nodes visited by the respective algorithm. Times are in seconds. The ILP was canceled after 300s of processing, the search-tree algorithms were canceled after 600s of processing. The column labeled "%" gives the percentage of instances solved in under 600 seconds. The figures in columns 5–8 have been measured in 2005 [182].

				running time (s)			search-tree size			
$\|V_i\|$	$\|E\|$	f	%	max	med	avg	min	max	med	avg
20	20	0	100	0.04	0.01	0.01	1	5	1	1
20	25	1	100	0.02	0.01	0.01	1	5	1	1
20	30	2	100	0.01	0.01	0.01	1	7	1	2
20	35	3	100	0.01	0.01	0.01	1	22	2	4
20	40	6	100	0.01	0.01	0.01	1	110	18	26
20	45	10	100	0.04	0.01	0.01	8	935	80	157
20	50	14	100	0.22	0.02	0.04	15	6,753	253	897
20	55	18	100	1.69	0.01	0.08	17	64,137	102	2,417
20	60	22	100	8.27	0.01	0.55	20	283,476	128	18,596
20	65	27	99		0.02	8.35	23	$> 9M$	117	305,501
20	70	31	98		0.02	14.21	32	$> 21M$	107	489,962
20	75	36	99		0.02	4.25	35	$> 8M$	85	147,080
20	79	41	99		0.01	0.73	33	$> 1M$	88	27,630
20	84	46	99		0.02	2.28	43	$> 3M$	139	82,563
20	89	51	99		0.02	2.30	44	$> 5M$	77	89,623
20	94	55	97		0.02	3.20	46	$> 5M$	84	126,372
20	99	60	96		0.02	1.07	56	$> 1M$	109	37,733
20	40	6	100	0.02	0.01	0.01	1	231	13	24
30	60	9	100	0.16	0.01	0.02	1	3,099	73	231
40	80	13	100	1.92	0.02	0.07	4	53,860	212	1,546
50	100	16	99		0.04	0.91	1	$> 1M$	403	21,754
60	120	19	99		0.22	1.90	1	$> 1M$	3,852	37,187
70	140	23	99		0.61	6.22	32	$> 1M$	11,098	112,015
80	160	26	90		1.90	21.65	158	$> 5M$	37,172	339,282
90	180	29	91		3.66	43.76	8	$> 6M$	60,600	661,619
100	200	32	81		26.16	73.52	132	$> 8M$	323,228	1,097,335

Table 2.2: Detailed results for running times and search-tree sizes of our $O^*(3.8^f)$-time algorithm (f denoting the feedback edge set number) run on the first batch of tests. The minimum running times never exceeded $10ms$ and were therefore dropped from the table. Since maxima are not very meaningfull if the timelimit of $600s$ was hit, maximum running times are omitted and for maximum search-tree sizes, we just give a lower bound. Herein, $> 6M$ means that a canceled process had explored over 6 million search-tree nodes at the point of termination. Note that averages also loose meaning in this case, but medians do not.

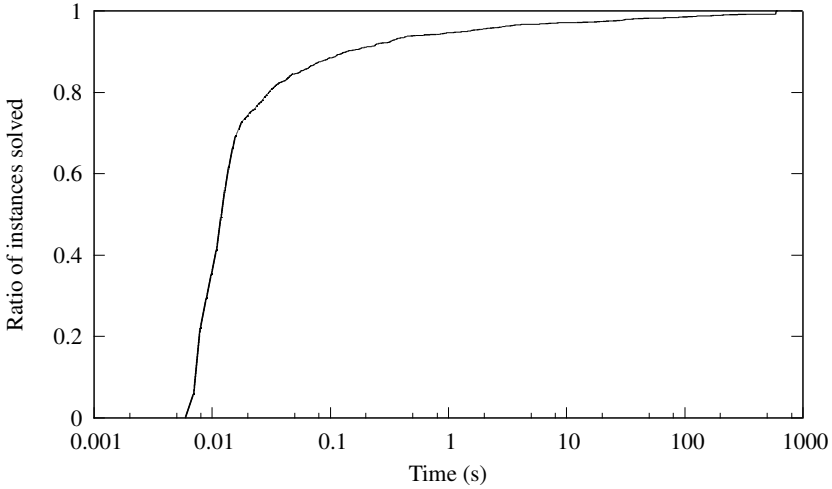


Figure 2.12: Cumulative distribution function of the running time of the $O(3.8^f f^2 + f|G|)$-time algorithm on "dense" graphs (first 17 rows of Table 2.1). It plots the percentage of instances solved in time at most x versus the time x. Note that the x-axis is in logscale.

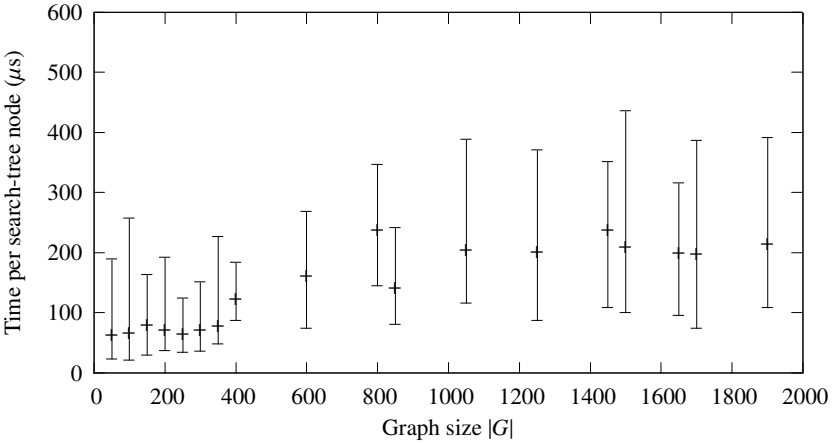


Figure 2.13: Average, minimum, and maximum time spent per search-tree node by our algorithm over all experiments. Only search trees with ≥ 10 search-tree nodes are considered.

					running time (s)			search-tree size			
$\|V\|$	p	f	k	#	max	median	avg	min	max	median	avg
100	12%	12	20	30	0.01	0.01	0.01	1	65	5	8
200	12%	24	40	30	0.06	0.01	0.01	1	506	9	50
300	12%	36	61	30	0.24	0.01	0.03	1	2,529	18	173
400	12%	48	81	30	0.06	0.01	0.02	1	327	8	53
500	12%	60	103	30	0.68	0.01	0.04	1	2,569	12	122
600	12%	72	123	30	0.28	0.01	0.03	1	1,234	7	123
700	12%	84	143	30	0.43	0.01	0.04	1	3,882	11	211
800	12%	96	164	30	5.55	0.01	0.37	1	28,684	3	2,013
900	12%	108	185	30	5.30	0.01	0.26	1	23,007	11	1,143
1000	12%	120	206	30	0.49	0.03	0.07	1	1,466	10	172
100	15%	15	22	30	0.06	0.01	0.01	1	873	22	72
200	15%	30	44	30	0.36	0.02	0.06	1	3,249	36	455
300	15%	45	66	30	4.23	0.04	0.43	1	40,287	174	4,475
400	15%	60	88	30	42.05	0.34	3.60	17	255,928	2,862	26,277
500	15%	75	111	30	409.56	0.39	21.38	1	2,774,889	2,592	143,482
600	15%	90	134	27		6.04	32.38	18	$> 2M$	41,900	194,055
700	15%	105	156	24		21.65	64.31	202	$> 2M$	112,614	317,224
800	15%	120	180	24		13.24	61.55	3	$> 2M$	95,136	347,082
900	15%	135	201	22		65.93	73.47	22	$> 2M$	249,645	368,608
1000	15%	150	225	16		109.78	20.54	104	$> 1M$	316,186	61,974

Table 2.3: Detailed results for running times and search-tree sizes of our $O(3.8^f f^2 + f|G|)$-time algorithm (f denoting the feedback edge set number) run on the second batch of tests. The instances consist of a random tree on the vertex set V augmented by $p \cdot |V|$ edges. The fifth column labeled "#" shows the number of instances that were solved within the timelimit of $600s$. For each row, 30 instances were generated. The minimum running times never exceeded $10ms$ and were therefore dropped from the table. Since maxima are not very meaningful if the timelimit of $600s$ was hit, maximum running times are omitted and for maximum search-tree sizes, we just give a lower bound. Herein, $> 2M$ means that a canceled process had explored over 2 million search-tree nodes at the point of termination. Note that averages also loose meaning in this case, but medians do not.

Chapter 3

Kernelization and Efficiency

A popular idea to simplify an input graph is replacing large parts of the input that have limited "interaction" with the rest of the graph by smaller parts, that mimic this interaction (that is, to the rest of the graph, it does not matter whether the large or the small part interacts with it). This concept has been formalized and examined extensively [18, 27, 92, 93, 104, 105, 121, 132]. A groundbreaking first result in this line of research was the linear-size problem kernel for Dominating Set on planar graphs [8]. This result can be seen as a catalyst for the rapid growth of results on problem kernels. In this chapter, again Dominating Set on planar graphs serves as a starting example, now for studying the issue of *linear-time* kernelizability as a natural goal within polynomial-time data reduction. To date, kernelization races were mainly about the size of the produced problem kernel. For example, the NP-hard Feedback Vertex Set problem (given an undirected graph and a positive integer k, find at most k vertices whose deletion destroys all cycles in G), was shown to admit an $O(k^{11})$-vertex problem kernel [33], which was later improved to an $O(k^3)$-vertex problem kernel [24] and finally to an $O(k^2)$-vertex problem kernel [188] (with unspecified polynomial running time). A similar story can be told for the arc-deletion variant Feedback Arc Set restricted to tournaments (that is, oriented cliques). Dom et al. [62] observed that a simple "high-degree" preprocessing yields a problem kernel containing $O(k^2)$ vertices. Then, Bessy et al. [16] improved this bound to a linear number of vertices using so-called transitive modules. Notably, modular decompositions have proven useful in developing kernelization algorithms for graph modification problems [103] and can be computed in linear time [108, 152]. In the case of Feedback Arc Set in Tournaments, however, additional time-consuming rules are required [16].

In some sense, the race for better and better bounds on the problem kernel size (by developing polynomial-time data reduction rules) goes in parallel to the race for better factors in polynomial-time approximation algorithms. From the viewpoint of practical relevance, however, the omission of running time considerations beyond mere "polynomial time" is a significant deficiency in these (theoretically well justified) races. In many practical settings even quadratic running times may be unacceptable [79, 171, 187]. Hence, it is a natural goal to see what solution quality, let it be approximation factor or problem kernel size, can be

achieved when requiring *linear-time* computability. For instance, returning to (the minimization version of) Feedback Vertex Set, there is a polynomial-time factor-2 approximation algorithm [12] and a linear-time factor-4 approximation algorithm [13]. Thus, it is natural to analyze the trade-off between output quality and running time. This is at least as interesting for kernelization algorithms as it is for approximation algorithms, since kernelizations can be combined with other algorithms. For example, combining a linear-time cubic-size kernelization with a cubic-time linear-size kernelization allows computation of a linear-time problem kernel in "almost linear" time (for small parameter values). Combining a linear-time kernelization with an approximation algorithm can improve the quality of the solution, since a smaller part of the solution needs to be approximated. This leads to the central question "what problem kernel sizes can be achieved in linear time?". Unfortunately, so far this question has been widely neglected in designing kernelization algorithms and only recently received elevated attention [19, 109, 110, 168].

In this chapter, we show that a linear-size problem kernel for Dominating Set on planar graphs can be computed in *linear* time, whereas previous kernelization algorithms needed cubic time [8, 46]. Since this turns out to be a demanding task, we refrain from analyzing the constant factor for the problem kernel size.[19] We affirmatively answer a question posed independently by Jiong Guo and Saket Saurabh at the Workshop on Kernelization 2010 (*WorKer'10*) held in November 2010 in Leiden, Netherlands; that is, a *linear-time* linear-size kernel for Dominating Set in planar graphs is possible.

Synchronously with our initial publication of large parts of this chapter, another linear-time kernelization for Dominating Set on planar graphs was shown by Hagerup [109]. While the new reduction rules presented by Hagerup [109] may lead to a less complex analysis, our approach of recycling the old reduction rules of Alber et al. [8] may have the advantage of greater versatility: our approach can likely be applied to a variety of NP-hard problems on planar graphs (see [104]) and bears the possibility of improving the kernel-size similarly as the work of Chen et al. [46] and Wang et al. [194].

Emphasizing and tuning of running times of preprocessing algorithms becomes increasingly popular. For example, van Bevern [18] presented a linear-time kernelization for the famous Hitting Set problem with constant-size sets. Linear-time kernelizations have also been shown for the Edge Dominating Set problem with respect to the solution size k [201, 110].

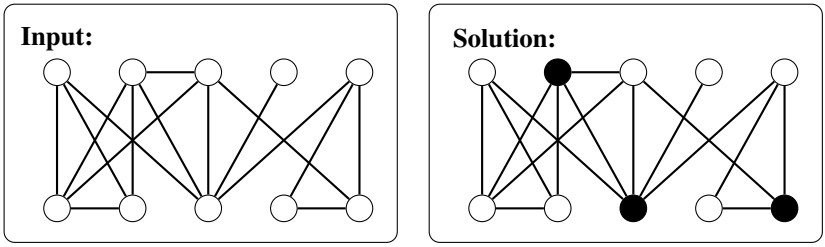
[19] A standard analysis leads to large constants—however, a refined analysis may improve them significantly.

3.1 Introduction to Dominating Set on Planar Graphs

Planar graph problems have played an important role in the development of several lines of research in parameterized complexity analysis. For example, the topic of subexponential time fixed-parameter algorithms was first studied for Dominating Set on planar graphs [7, 64].

Intuitively, Dominating Set is the problem of finding a small "dominating set" in a graph. Herein, a *dominating set* is a set of vertices such that all other vertices have a neighbor in the dominating set. More formally, we define the following.

k-Dominating Set
Input: An undirected graph $G = (V, E)$ and an integer $k \geq 0$.
Question: Is there a vertex set $V' \subseteq V$ with $|V'| \leq k$ and $N[V'] = V$?
Parameter: k



It is interesting to note that, for our considerations, knowledge of the parameter value k, that is, the maximum allowed size of a dominating set, will not be explicitly used in our algorithms. Instead, we formulate our results with respect to the "domination number $\gamma(G)$ of the input graph G", that is, the smallest number γ such that $(G, \gamma) \in$ Dominating Set.

In this chapter, we consider Dominating Set restricted to planar input graphs. For this variant, there was at first a $335k$-vertex problem kernel [8], which was later refined into a $67k$-vertex problem kernel [46], both computable in cubic time.[20] Chen et al. [46] focused on "engineering" data reduction rules in order to obtain a small provable kernel size. Here, we aim at engineering the usage of known (and "established") data reduction rules to improve the time complexity, instead of aiming for new data reduction rules.

[20] Experimental work showed that the corresponding data reduction rules are useful on several real-world data sets [9].

Our Contributions

Revisiting previous data reduction rules for Dominating Set on planar graphs [8], we shift the focus to the execution time of data reduction, improving it from cubic to linear time. To this end, we "rework" the known rules and their mathematical analysis and carefully analyze their interaction. Our central observation is that one can significantly gain efficiency by a non-exhaustive application of data reduction rules. More specifically, implementing the known data reduction rules [8] in the natural and straightforward way would "unavoidably" lead to quadratic running time: The reason for this is that one has to inspect a quadratic number of vertex pairs that all may define a so-called (potential) region of a planar embedding of the input graph. In fact, Alber et al. [8] proved that their reduction rules can be implemented to run in cubic time. Thus, one of our major technical contributions is to restrict the region decomposition concept (introduced by Alber et al. [8]) so that the inspection and, thus, the data reduction can be done much faster. In this way, we achieve an $O(k)$-vertex problem kernel for Dominating Set on planar graphs in linear time. Our main theorem is the following.

Theorem 3.1. Dominating Set *on planar graphs admits a problem kernel of size* $O(\gamma(G))$. *This kernel can be computed in linear time.*

Notably, our kernel size analysis is not as fine-grained as the previous ones [8, 46], meaning that we did not analyze the constant factor for the upper bound on the number of problem kernel vertices. However, since multiple kernelization algorithms can be run on top of each other (this makes them quite different from approximation algorithms), using our algorithm as spear-head in combination with Chen et al.'s algorithm [46], we can achieve a problem kernel with $67k$ vertices in $O(n + k^3)$ time. This makes the quest for a sharper kernel size analysis somewhat moot.

Organization of the Chapter

In Section 3.2, we recall the basic ideas of Alber et al. [8] and compare their results to ours. In Section 3.3, we present our data reduction rules and show their correctness and running time. We go on to show how these rules are to be executed in order to achieve the claimed kernelization in Section 3.4. We conclude the chapter in Section 3.5 with a short resume and some open questions in the context of linear-time problem kernelization. To help navigate the results, Figure 3.1 gives an overview over the dependency graph of the lemmas and propositions used in this chapter.

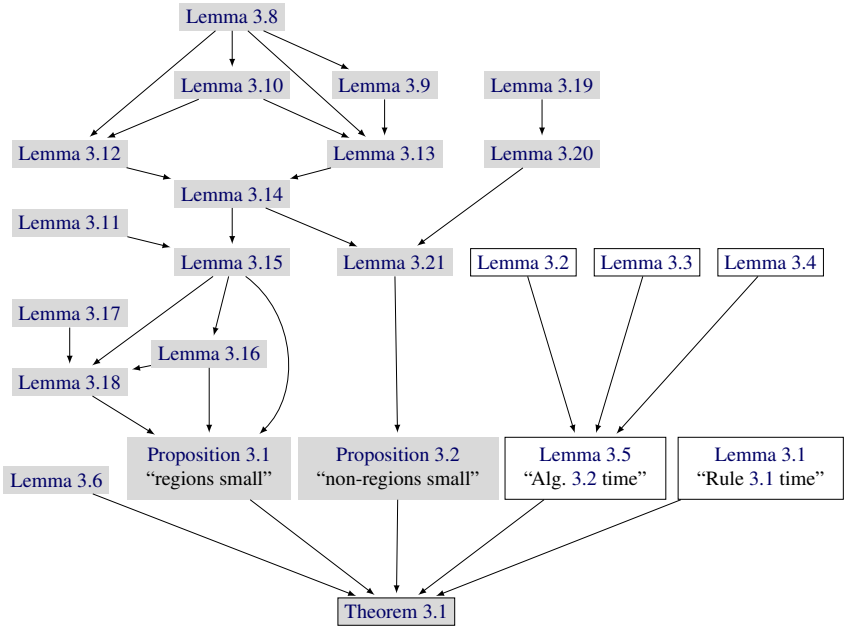


Figure 3.1: The structure of statements used to prove Theorem 3.1, the main theorem of this chapter. Statements with a gray background are for bounding the size of the kernel while statements with a black border help bound the running time of the kernelization. As seen in this picture, we show the running time in the earlier sections and prove the kernel size later.

Notation

In our algorithms, graphs are represented as adjacency lists. We use the *joint neighborhood* $N^G(v, w)$ of two vertices to denote $(N^G(v) \cup N^G(w)) \setminus \{v, w\}$ and the *closed joint neighborhood* $N^G[v, w] := N^G[v] \cup N^G[w]$. A set $U \subseteq V(G)$ *separates* a vertex v from a set $U' \subseteq V(G)$ if every path from v to a vertex in U' contains a vertex of U.

A K_5 is a clique with five vertices. A $K_{3,3}$ is a graph with vertex set $V \uplus V'$, where each vertex in V is adjacent to every vertex in V' and $|V| = |V'| = 3$. A graph G' is a *minor* of G if G can be transformed into G' by edge deletions, vertex deletions, and edge contractions. By the famous Wagner-Kuratowski theorem, a graph G is planar if and only if neither the K_5 nor the $K_{3,3}$ is a minor of G. A planar graph can be *embedded* into the two-dimensional plane. A planar graph with a fixed

embedding is called a *plane graph*.
Note that planar graphs are sparse in the sense that the number m of edges is linear in the number n of vertices, more precisely $m \leq 3n - 6$ by Euler's polyhedron formula that has been shown many times in the literature [77]. and planarity is inherited by subgraphs (that is, deleting vertices or edges does not destroy the property of being planar).

3.2 Comparison to Previous Kernelizations

To obtain a linear-size problem kernel for Dominating Set on planar graphs, we employ a framework developed by Alber et al. [8]. They showed that a planar graph G with domination number $\gamma(G)$ can be decomposed into $O(\gamma(G))$ so-called "regions". By applying data reduction rules, they ensure that each of these regions has constant size and that only $O(\gamma(G))$ vertices are not contained in any region. We follow a similar approach, modifying their data reduction rules to run in linear time.
The notion of "regions" is central in this chapter. We give a short intuition of its definition. Given an embedding of a planar graph G, a region $\mathcal{R}$ can be pictured as an area of this embedding that is enclosed by two paths p_1 and p_2 of length at most three. $\mathcal{R}$ can therefore *contain* vertices, as illustrated in Figure 3.2. Each region contains two vertices v and w such that $N[v, w]$ contains all vertices of $\mathcal{R}$ and a boundary (consisting of p_1 and p_2) that separates $\mathcal{R}$ from the rest of the plane. Furthermore, a region decomposition of an embedding of G is a collection of regions in this embedding such that no two regions overlap (two regions may touch each other at their boundary, sharing vertices of the two boundary paths). More formally, we define the following.

Definition 3.1. *Let G be a plane graph. A* region $\mathcal{R}(v, w)$ *between two vertices v and w ($v \neq w$)is a closed bounded subset of the plane such that:*

1. *the boundary of $\mathcal{R}(v, w)$ is formed by two v-w-paths[21], each of which has length at most three and*
2. *all vertices in $\mathcal{R}(v, w)$ are also in $N[v, w]$.*

For ease of presentation, we denote by $\mathcal{R}(v, w)$ also the set of vertices in a region $\mathcal{R}(v, w)$ and by $\partial\mathcal{R}(v, w)$ the set of vertices on the boundary paths of a region $\mathcal{R}(v, w)$ including v and w. The vertices in $\mathcal{R}(v, w) \setminus \partial\mathcal{R}(v, w)$ are the *inner vertices* of $\mathcal{R}(v, w)$.
Alber et al. [8] showed that each dominating set D of a planar graph G yields a so-called maximal D-region decomposition with $O(|D|)$ regions.

[21] This includes degenerated cases where the two paths have common vertices besides v and w.

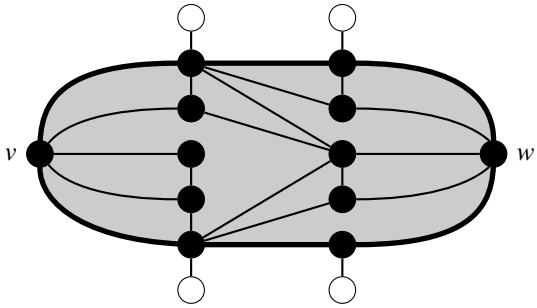


Figure 3.2: Illustration of a region $\mathcal{R}(v, w)$. The boundary $\partial\mathcal{R}(v, w)$ (consisting of two v-w-paths) is drawn with bold edges. The vertices in $\mathcal{R}(v, w)$ are drawn black.

Definition 3.2. *For a plane graph G and $D \subseteq V(G)$, a* D-region decomposition *of G is a set $\mathfrak{R}$ of regions between pairs of vertices in D such that*

1. *$\forall v, w \in D$ and $\mathcal{R}(v, w) \in \mathfrak{R}$, it holds that $D \cap \mathcal{R}(v, w) = \{v, w\}$ and*
2. *for two distinct regions $\mathcal{R}_1, \mathcal{R}_2 \in \mathfrak{R}$, it holds that $(\mathcal{R}_1 \cap \mathcal{R}_2) \subseteq (\partial\mathcal{R}_1 \cap \partial\mathcal{R}_2)$, that is, each vertex that is in both $\mathcal{R}_1$ and $\mathcal{R}_2$ is also part of the boundary of $\mathcal{R}_1$ and the boundary of $\mathcal{R}_2$.*

For a D-region decomposition $\mathfrak{R}$, we denote the set of vertices that are contained in regions of $\mathfrak{R}$ by $V(\mathfrak{R}) := \bigcup_{\mathcal{R}\in\mathfrak{R}} \mathcal{R}$. A D-region decomposition $\mathfrak{R}$ is maximal *if there is no region $\mathcal{R} \notin \mathfrak{R}$ such that $\mathfrak{R}' := \mathfrak{R} \cup \{\mathcal{R}\}$ is a D-region decomposition with $V(\mathfrak{R}) \subsetneq V(\mathfrak{R}')$.*

Using data reduction, Alber et al. [8] shrink to constant size all regions that may potentially be part of a D-region decomposition for a minimum dominating set D of the input graph G. Since $|D| = \gamma(G)$, such a region decomposition comprises $O(\gamma(G))$ regions. Together with an $O(\gamma(G))$-bound on the number of vertices that are not in regions, this shows the linear size of the kernel. This data reduction can be performed in $O(n^3)$ time [8].
Our goal is to modify the data reduction rules of Alber et al. [8] so that they can be applied in linear time instead of cubic time. Unfortunately, we have to make some sacrifices regarding the effectiveness of the data reduction rules, which we explain in Section 3.3.

3.3 Data Reduction Rules

In this section, we first describe two data reduction rules and show that they are *correct*, that is, they maintain planarity and do not change the domination number

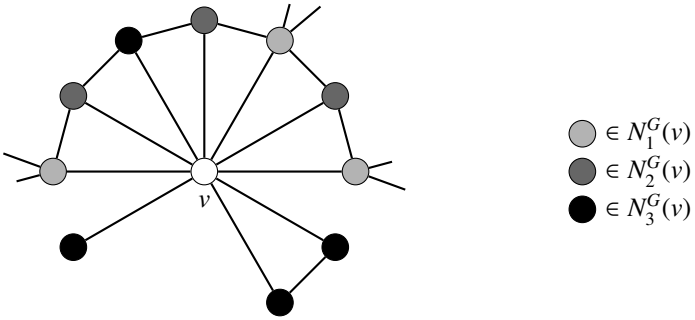


Figure 3.3: Illustration of the sets $N_1^G(v)$ ("exits"), $N_2^G(v)$ ("guards"), and $N_3^G(v)$ ("prisoners").

of the input graph. Then, we show how to execute them in linear time. Whenever we introduce new vertices into a graph, we call them *dummy vertices*. Moreover, we assume that our data reduction rules can check in O(1) time whether a vertex is a dummy vertex. This can be achieved by marking dummy vertices accordingly. Note that these marks are to be removed from the final output graph in order to obtain a proper Dominating Set instance (where unmarked graphs are required as input).

3.3.1 Private Neighborhood Rule

Following Alber et al. [8], we partition the neighborhood of a vertex v of a graph G into three subsets (see Figure 3.3): "exits" are neighbors of v who have another neighbor outside the neighborhood of v. "Guards" are neighbors of exits of v. All remaining neighbors of v are called "prisoners". More formally, we define the following.

$$\begin{aligned}
\text{"exits"}\ N_1^G(v) &:= \{u \in N^G(v) : N^G(u) \setminus N^G[v] \neq \emptyset\},\\
\text{"guards"}\ N_2^G(v) &:= \{u \in N^G(v) \setminus N_1^G(v) : N^G(u) \cap N_1^G(v) \neq \emptyset\},\\
\text{"prisoners"}\ N_3^G(v) &:= N^G(v) \setminus (N_1^G(v) \cup N_2^G(v)).
\end{aligned}$$

For simplicity, we use $N_{1,2}^G(v)$ as shorthand for $N_1^G(v) \cup N_2^G(v)$ and, likewise, $N_{1,3}^G(v)$ and $N_{2,3}^G(v)$ for $N_1^G(v) \cup N_3^G(v)$ and $N_2^G(v) \cup N_3^G(v)$, respectively.
Now, we can formulate our variant of Rule 1 of Alber et al. [8] for planar graphs. The conditions on the neighborhood-sizes ensure that the reduction rule is not applicable if $N_3^G(v)$ consists of a single degree-one vertex.

Reduction Rule 3.1. *Let $v \in V(G)$ and let $|N_3^G(v)| > 1$ or $|N^G(N_3^G(v))| > 1$. Then, remove $N_3^G(v)$ from G and attach a new degree-one dummy vertex v' to v.*

The decisive distinction between our reduction rule and the original one of Alber et al. [8] is that we do not delete $N_2^G(v)$. This helps us exclude cascades of Reduction Rule 3.1. The conditions on the neighborhood-sizes ensure that the reduction rule is not applicable if $N_3^G(v)$ consists of a single degree-one vertex.
Intuitively, a prisoner or guard of v cannot dominate more than v and since all prisoners have to be dominated, v can be assumed to be in an optimal solution. Formally, the correctness of Reduction Rule 3.1 follows from the correctness of Reduction Rule 1 used by Alber et al. [8], as we only delete a subset of the vertices that Alber et al. show to be safely removable.
Before we show the running-time of Reduction Rule 3.1, observe that, whenever it is applied to a vertex v satisfying the conditions of Reduction Rule 3.1, v gains a degree-one-neighbor. Thus, we can observe that v is neither a guard nor a prisoner for (almost) any vertex

Observation 3.1. *Let v be a vertex whose neighborhood contains a degree-one-vertex w. Then, for all $u \in N^G(v) \setminus \{w\}$, it holds that $v \notin N_{2,3}^G(u)$ and, therefore, $N^G[v] \cap N_3^G(u) = \emptyset$.*

By removing only a subset of the removable vertices, we can show that, for an exhaustive application of Reduction Rule 3.1, it is sufficient to apply Reduction Rule 3.1 once for every vertex. In this way, we can prove Lemma 3.1.

Lemma 3.1. *For planar graphs, Reduction Rule 3.1 can be applied exhaustively in $O(n)$ time.*

Proof. Let G be a planar graph, and let $v \in V(G)$ be a vertex that does not satisfy the conditions of Reduction Rule 3.1, that is, neither $|N_3^G(v)| > 1$ nor $|N^G(N_3^G(v))| > 1$. We show that, in the graph G' that results from applying Reduction Rule 3.1 to a vertex $u \in V(G) \setminus \{v\}$, the vertex v still does not satisfy the conditions of Reduction Rule 3.1. This implies that, in order to apply Reduction Rule 3.1 exhaustively to G, it is sufficient to apply Reduction Rule 3.1 at most once to each vertex. As shown by Alber et al. [8, Lemma 2], this can be done in $O(n)$ time for planar graphs.
Towards a contradiction, assume that Reduction Rule 3.1 is applicable to v in G'. Then, because Reduction Rule 3.1 does not add edges between vertices in $V(G)$, all prisoners of v in G' are neighbors of v in G. However, since Reduction Rule 3.1 is not applicable to v in G, it holds that $N_3^{G'}(v) \cap N_{1,2}^G(v) \neq \emptyset$. In the following, we show that $N_3^{G'}(v)$ and $N_{1,2}^G(v)$ are disjoint, contradicting the assumption that Reduction Rule 3.1 is applicable to v in G'. To this end, recall that each vertex in $N_2^G(v)$

is adjacent to a vertex in $N_1^G(v)$. Hence, it is sufficient to show that, for each $x \in N_1^G(v)$, it holds that $N^G[x] \cap N_3^{G'}(v) = \emptyset$. In the following, we show $N^G[x] \subseteq N^G[u]$. This is sufficient since, by Observation 3.1, $N^{G'}[u] \cap N_3^{G'}(v) = \emptyset$, implying

$$N^G[x] \cap N_3^{G'}(u) = N^G[x] \cap (N_3^{G'}(u) \cap V(G')) \subseteq N^{G'}[u] \cap N_3^{G'}(u) = \emptyset.$$

We distinguish the following cases:

Case 1: $x \in N_{2,3}^{G'}(v)$. Then, Reduction Rule 3.1, when applied to u, deletes all neighbors of x that are nonadjacent to v. Let $y \in N^G(x) \setminus N^G[v]$ be one such neighbor. Since y is deleted by Reduction Rule 3.1, we know that $y \in N_3^G(u)$, implying $x \in N^G[u] \setminus N_1^G(u)$. Then, however, $N^G[x] \subseteq N^G[u]$, implying $N^{G'}[x] \subseteq N^{G'}[u]$. Hence, u is adjacent to v in G. Because u has a degree-one dummy neighbor in G', no vertex in $N^{G'}[u]$ is contained in $N_3^{G'}(v)$. Since $N^G(x) \cap V(G') \subseteq N^{G'}[u]$, this implies $N^G[x] \cap N_3^{G'}(v) = \emptyset$.

Case 2: $x \notin N_{2,3}^{G'}(v)$. Then, it suffices to show that the vertices in $N^G(x)$ are not in $N_3^{G'}(v)$. If $x \in N_1^{G'}(v)$, then $N^G(x) \cap N_3^{G'}(v) = \emptyset$. Hence, consider the subcase where $x \notin N_1^{G'}(v)$. This implies $x \notin N^{G'}(v)$ and, thus, x would have been deleted by Reduction Rule 3.1. Therefore, we have $x \in N_3^G(u)$, that is, $N^G[x] \subseteq N^G[u]$, implying $v \in N^G(u)$. Again, because u has a degree-one dummy neighbor in G', no vertex in $N^{G'}[u]$ is contained in $N_3^{G'}(v)$, implying $N^G[x] \cap N_3^{G'}(v) = \emptyset$. □

In the following, we say that a graph G is *reduced* with respect to Reduction Rule 3.1 if Reduction Rule 3.1 is not applicable to G.

3.3.2 Joint Neighborhood Rule

In this section, we present a data reduction rule that shrinks regions to constant size. This rule is based on Reduction Rule 2 by Alber et al. [8] which removes certain vertices from the sets $N(v, w)$ for vertices $v, w \in V$. However, we cannot compute $N(v, w)$ for all vertex pairs $v, w \in V$ in linear time. We circumvent this problem by showing that it is sufficient to only remove vertices from efficiently-computable subsets $N_0(v, w) \subseteq N(v, w)$ for a linear number of vertex pairs. More specifically, $N_0(v, w)$ contains vertices on short low-degree v-w-paths:

Definition 3.3. *A vertex v with $\deg(v) \leq 78$ is called a* low-degree vertex. *A v-w-path consisting only of v, w, and low-degree vertices is called a* low-degree path.

The constant 78 as bound for low-degree vertices is the result of the mathematical analysis and can almost surely be improved using a more intricate analysis.

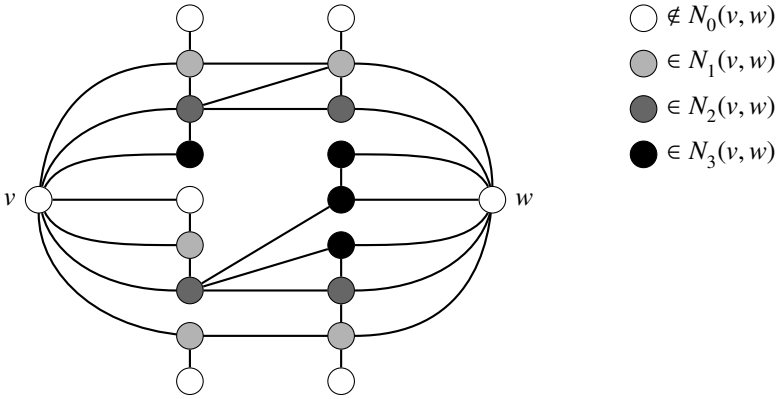


Figure 3.4: An illustration of Definition 3.4.

Note that all low-degree paths of constant length c starting at some vertex v can be found in $O(\deg(v))$ time by starting a breadth-first search at v, only descending on low-degree vertices and stopping at depth c.

Observation 3.2. *For a vertex v, all constant-length low-degree paths starting at v can be listed in $O(\deg(v))$ time.*

To present our data reduction rule, we need the following definition of joint neighborhoods. It strongly resembles the definition used by Alber et al. [8, Section 2.2]. The difference here is that our sets $N_i^G(v,w)$ for $i \in \{1,2,3\}$ are defined with respect to $N_0^G(v,w)$ instead of $N^G(v,w)$. The set $N_0^G(v,w)$ consists of those vertices in $N^G(v,w)$ that are low-degree vertices and lie on a short low-degree v-w-path. These two properties allow us to compute $N_0^G(v,w)$ efficiently. The following definition is illustrated in Figure 3.4.

Definition 3.4. *Let v, w be vertices in a planar graph G. We define*

$$N_0^G(v,w) := \{u \in N^G(v,w) : u \text{ is on a low-degree } v\text{-}w\text{-path of length at most four that consists entirely of vertices in } N^G[v,w]\},$$
$$N_0^G[v,w] := N_0^G(v,w) \cup \{v,w\}$$
$$\text{“exits” } N_1^G(v,w) := \{u \in N_0^G(v,w) : N^G(u) \setminus N_0^G[v,w] \neq \emptyset\},$$
$$\text{“guards” } N_2^G(v,w) := \{u \in N_0^G(v,w) \setminus N_1^G(v,w) : N^G(u) \cap N_1^G(v,w) \neq \emptyset\},$$
$$\text{“prisoners” } N_3^G(v,w) := N_0^G(v,w) \setminus (N_1^G(v,w) \cup N_2^G(v,w)).$$

Again, we write $N^G_{1,2}(v,w)$, $N^G_{1,3}(v,w)$ *and* $N^G_{2,3}(v,w)$ *for* $N^G_1(v,w) \cup N^G_2(v,w)$, $N^G_1(v,w) \cup N^G_3(v,w)$ *and* $N^G_2(v,w) \cup N^G_3(v,w)$, *respectively.*

Using Definition 3.4, we present our second data reduction rule in form of Algorithm 3.2, which we then explain. Algorithm 3.2 basically corresponds to Reduction Rule 2 of Alber et al. [8] with the main difference that we do not explore the whole neighborhood $N(v) \cup N(w)$ of v and w but rather a large enough, efficiently computable part of it. We also make multiple small adjustments in order to avoid having to reapply Reduction Rule 3.1 to v or w. For each pair $v, w \in V$ with $N^G_0(v,w) \neq \emptyset$, Algorithm 3.2 removes a subset of $N^G_0(v,w)$ from the graph G and, if applicable, attaches degree-one dummy vertices to v or w. We first explain the data reduction between lines 6 and 11 and then explain the purpose of the EnsurePaths procedure called in lines 4 and 5. The set N_3 introduced in line 6 of Algorithm 3.2 is the set of vertices that may possibly be removed. We will see that N_3 can be efficiently computed and updated. Moreover, the choice of N_3 ensures the correctness of the data reduction executed by Algorithm 3.2, which can be seen by comparing it to Reduction Rule 2 of Alber et al.: it is straightforward to observe that, if the condition in line 7 is satisfied, then the corresponding condition for Reduction Rule 2 of Alber et al. is also satisfied. Moreover, in this case, we remove only vertices that Alber et al. showed to be safely removable. Other than Alber et al. [8], we choose vertices z, z' in line 8 that *already exist* in the graph. For us, this is important to be able to state that only (degree-one) dummy vertices are ever added by the reduction rules and that no dummy vertex is attached to other dummy vertices, which will simplify later proofs to some extend.

Lemma 3.2. *Let* v, w *be vertices of a planar graph* G' *and let* $N_3 \subseteq N^{G'}_3(v,w) \cap N^{G'}(v) \cap N^{G'}(w)$ *with* $|N_3| \geq 4$. *Then,* N_3 *contains vertices* z, z' *with* $\{z, z'\} \notin E(G')$ *and* $(N^{G'}(z) \cap N^{G'}(z')) \setminus N_3 = \{v, w\}$.

Proof. First, observe that a vertex $u \in V(G') \setminus \{v, w\}$ cannot have three distinct neighbors $u_1, u_2, u_3 \in N_3$ as this implies a $K_{3,3}$-minor on the vertex set $\{u_1, u_2, u_3\} \uplus \{u, v, w\}$ in G', contradicting its planarity. More specifically, this implies that, for each $z, z' \in N_3$, a vertex in $(N^{G'}(z) \cap N^{G'}(z')) \setminus \{v, w\}$ does not have a neighbor other than z, z' in N_3.

Assume towards a contradiction that, for any two vertices $z, z' \in N_3$, it holds that $\{z, z'\} \in E(G')$ or there is a vertex $y_{zz'} \in (N^{G'}(z) \cap N^{G'}(z')) \setminus (N_3 \cup \{v, w\})$. In order to show that the vertices in $N_3 \cup \{v\}$ form a K_5 minor, which contradicts the planarity of G', consider a pair of vertices $z, z' \in N_3$ with $\{z, z'\} \notin E(G')$. By assumption, there is a vertex $y_{zz'} \in (N^{G'}(z) \cap N^{G'}(z')) \setminus (N_3 \cup \{v, w\})$ and, by the observation above, $N^{G'}(y_{zz'}) \cap N_3 = \{z, z'\}$. Thus, contracting one of the edges $\{y_{zz'}, z\}$ and $\{y_{zz'}, z\}$ for all such pairs $z, z' \in N_3$, for which a vertex $y_{zz'}$ exists,

Algorithm 3.2: Reduce Vertices in Regions.

1 compute $N_0^G(v, w)$ for all $v, w \in V$ with $N_0^G(v, w) \neq \emptyset$;
2 $G' \leftarrow G$;
3 **foreach** (v, w) *such that* v *and* w *are non-dummy vertices and* $N_0^G(v, w) \neq \emptyset$ **do**
4 EnsurePaths($G', v, N_0^G(v, w)$);
5 EnsurePaths($G', w, N_0^G(v, w)$);
6 $N_3 \leftarrow N_3^{G'}(v, w) \cap N_0^G(v, w)$;
7 **if** $|N_3| \geq 4$ *and no vertex* $u \notin \{v, w\}$ *dominates* N_3 *in* G' **then**
8 **if** $N_3 \subseteq N^{G'}(v) \cap N^{G'}(w)$ **then** remove the vertices in $N_3 \setminus \{z, z'\}$ from G', for arbitrary $z, z' \in N_3$ with $(N^{G'}(z) \cap N^{G'}(z')) \setminus N_3 = \{v, w\}$ and $\{z, z'\} \notin E(G')$;
9 **else if** $N_3 \subseteq N^{G'}(v)$ *and* $N_3 \not\subseteq N^{G'}(w)$ **then** remove N_3 from G' and (unless already done before) attach a degree-one dummy vertex v' to v;
10 **else if** $N_3 \not\subseteq N^{G'}(v)$ *and* $N_3 \subseteq N^{G'}(w)$ **then** remove N_3 from G' and (unless already done before) attach a degree-one dummy vertex w' to w;
11 **else if** $N_3 \not\subseteq N^{G'}(v)$ *and* $N_3 \not\subseteq N^{G'}(w)$ **then** remove N_3 from G' and (unless already done before) attach a degree-one dummy vertex v' to v and a new degree-one dummy vertex w' to w;
12 **return** G';

results in a minor of G' where all vertices in N_3 are pairwise adjacent. Because all vertices in N_3 are adjacent to v and since $|N_3| \geq 4$, the vertices in N_3 together with v form a forbidden K_5 minor. □

We now explain the EnsurePaths procedure called in line 4 and line 5 for a vertex pair (v, w). Observe that the graph G' considered in the for-loop in line 3 of Algorithm 3.2 is not necessarily reduced with respect to Reduction Rule 3.1 since previous iterations of the loop might have deleted vertices. Thus, it might happen that some vertex $u \in N_0^G(v, w)$ is not in $N_0^{G'}(v, w)$ when the pair (v, w) is considered in line 3. Such a situation is illustrated in Figure 3.5 and could arise if all of u's low-degree v-w-paths are destroyed by data reduction rules executed for some other vertex pair. As illustrated in Figure 3.5, this could prevent Algorithm 3.2 from removing u or some of its neighbors. In the situation shown, applying Reduction Rule 3.1 to v would delete u. Hence, in order to ensure that a vertex in G' that is in $N_0^G(v, w)$ in on some low-degree v-w-path in G', it could help to

Procedure EnsurePaths($G', x, N_0^G(v,w)$)

// for $x, u \in V(G')$, let $B(x,u) := \{u' \in V(G') \setminus \{x\} : \text{dist}_{G'-\{x\}}(u,u') \leq 2\}$

1 $N_3^{\text{ldeg}}(x) \leftarrow \{u \in N_3^{G'}(x) \cap N_0^G(v,w) : B(x,u)$ has only low-degree vertices$\}$;

2 **if** $|N^{G'}(N_3^{\text{ldeg}}(x))| > 1$ **then** remove $N_3^{\text{ldeg}}(x)$ from G' and (unless x already has one) attach a new degree-one dummy vertex to x;

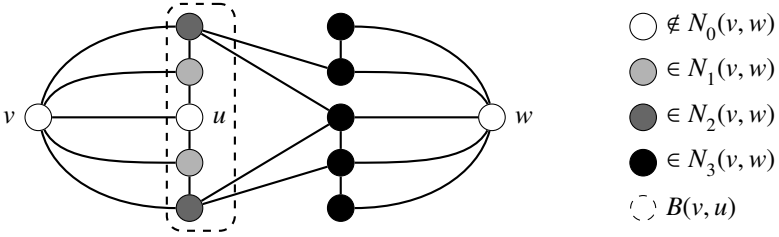


Figure 3.5: In G', the vertex u is not on a v-w-path of length at most four. Therefore, u and its neighbors are not deleted by a call of Algorithm 3.2. However, $u \in N_3(v)$. Hence, Reduction Rule 3.1 would delete u. Recall that $B(v,u)$ consists of all vertices that have distance at most two to u in the graph that results from deleting v. Since $B(v,u)$ (surrounded by the dashed line) contains only low-degree vertices, EnsurePaths will delete u as well.

apply Reduction Rule 3.1 to v and to w. However, doing so for each considered pair (v,w) might be too time-consuming.
For this reason, we employ EnsurePaths. Called for an $x \in \{v,w\}$, the procedure first computes a set of low-degree prisoners of x whose low-degree v-w-path has been destroyed. Then, it removes these vertices if applying Reduction Rule 3.1 would also delete them. The condition that $B(x,u)$ (that is, roughly the vertices at distance at most two from u in $N^{G'}(x)$) contains only low-degree vertices merely ensures that we can efficiently[22] check whether $u \in N_3^{G'}(x)$. This is needed since spending $O(\deg(x))$ time (as we did in Reduction Rule 3.1) for each region with anchor x may exceed linear time. Since EnsurePaths deletes only a subset of the vertices that Reduction Rule 3.1 would delete, EnsurePaths is correct. Moreover, observe that in Figure 3.5, an application of EnsurePaths to v would delete u.

To execute Algorithm 3.2 one can compute all sets $N_0^G(v,w)$ in linear time using Algorithm 3.3. Also, to compute N_3 in line 6, we have to verify containment in

[22] In this context, "efficiently" means "in overall linear time", that is, summing up the time spend in EnsurePaths over all regions with anchor x in a fix region-decomposition must not exceed $O(\deg(x))$.

$N_3^{G'}(v, w)$ for a number of vertices. The following lemma shows that this can be done efficiently.

Lemma 3.3. *Let G be a planar graph containing vertices u,v, and w. Then, $u \in N_3^G(v, w)$ can be checked in* $O(1)$ *time.*

Proof. To check whether some vertex u is in $N_3^G(v, w)$, we have to test $u \in N_0^G(v, w)$. This is equivalent to checking whether u is a low-degree vertex and whether u is on a low-degree v-w-path consisting of vertices in $N^G[v, w]$. Checking whether u is a low-degree vertex is possible in $O(1)$ time. If this is the case, then, according to Observation 3.2, generate a list L of all low-degree paths of length at most three starting at u in $O(\deg(u)) = O(1)$ time. We have to check whether two of these paths can be combined to a low-degree v-w-path in $N^G[v, w]$. Obviously, L contains $O(1)$ paths. Now, for the $O(1)$ pairs of paths $p, p' \in L$, check i) whether p and p' only intersect in u, ii) whether the lengths of p and p' sum up to at most four, iii) whether p connects v to u and p' connects u to w, and iv) whether all vertices in p and p' are in $N^G[v, w]$. Since all vertices except the endpoints of paths in L are low-degree vertices and since these paths have $O(1)$ length, these tests can be performed in $O(1)$ time. Obviously, $u \in N_0^G(v, w)$, that is, u is on a low-degree v-w-path in $N^G[v, w]$ of length at most four, if and only if a pair of paths p, p' passes all these tests.
Having shown how to decide $u \in N_0^G(v, w)$ in $O(1)$ time, we can now decide $u \in N_3^G(v, w)$. To this end, we first check $u \in N_0^G(v, w)$. If so, in $O(\deg(u)) = O(1))$ time, we enumerate all low-degree paths of length at most two that start in u and avoid $\{v, w\}$. There are $O(1)$ such paths. For each such path, we check in $O(1)$ time whether all of its vertices are in $N_0^G(v, w)$. This is the case for all paths if and only if $u \in N_3^G(v, w)$. □

Lemma 3.4. *The sets $N_0^G(v, w)$ for all vertices $v, w \in V(G)$ with $N_0^G(v, w) \neq \emptyset$ can be enumerated in* $O(n)$ *time.*

Proof. To prove the claim, Algorithm 3.3 is employed. The correctness of the computation is straightforward to verify, it remains to analyze Algorithm 3.3's running time. By Observation 3.2, the for-loop in line 2 runs $O(\deg(v))$ times for each $v \in V$. Hence, it runs at most $\sum_{v \in V} O(\deg(v) + 1) = O(n)$ times. Since we are dealing with low-degree paths of length $O(1)$, the body of the for-loop in line 2 takes $O(1)$ time. It follows that the for-loop in line 2 runs in $O(n)$ time and, furthermore, that $\mathcal{D}$ has $O(n)$ list entries. In line 6, the list $\mathcal{D}$ is sorted using radix sort. The keys to sort are triples. Hence, over the alphabet V, the length of the keys to sort is three. It follows that the sort in line 6 runs in $O(n)$ time. The for-loop in line 7 iterates over the $O(n)$ entries in $\mathcal{D}$, where each operation in the body clearly takes $O(1)$ time. Hence, Algorithm 3.3 runs in $O(n)$ time. □

Algorithm 3.3: Compute $N_0^G(v,w)$ for all $v,w \in V(G)$ with $N_0^G(v,w) \neq \emptyset$.

Input: A planar graph $G = (V,E)$ with vertices numbered $V := \{1,\dots,n\}$.
Output: $N_0^G(v,w)$ for all $v,w \in V$ with $N_0^G(v,w) \neq \emptyset$.

```
1  D ← empty list;   /* (v,w,u) ∈ D will be equivalent to u ∈ N_0^G(v,w) */
2  for v ∈ V and each low-degree path p of length at most four starting at v do
3      w ← ending vertex of p;
4      if all vertices of p are in N^G[v,w] then
5          foreach vertex u ∈ V \ {v,w} of p do append (v,w,u) to D;
6  sort D in lexicographic order using radix sort;
7  foreach (v,w,u) ∈ D in lexicographic order do   /* collect N_0^G(v,w) */
8      (v',w',u') ← previous element in D;
9      if v ≠ v' ∨ w ≠ w' then   /* first encounter of the pair (v,w) */
10         new set N_0^G(v,w) ← {u};
11     else if u ≠ u' then add u to N_0^G(v,w);   /* avoids duplicates */
12 return N_0^G(v,w) for all v,w ∈ V with N_0^G(v,w) ≠ ∅;
```

Since we can efficiently check membership of a vertex in $N_3^G(v,w)$ and we can compute the sets $N_0^G(v,w)$ for all pairs of vertices v and w in linear time (see Lemma 3.4), we have all the ingredients needed to prove the running time of Algorithm 3.2.

Lemma 3.5. *On planar graphs, Algorithm 3.2 can be executed in* $O(n)$ *time.*

Proof. We have to show that Algorithm 3.2 works in $O(n)$ time. By Lemma 3.4, line 1 of Algorithm 3.2 can be executed in $O(n)$ time. We assume that line 1 yields a list $\mathcal{L}$ of pairs (v,w) with $N_0^G(v,w) \neq \emptyset$. Also, whenever a new degree-one dummy vertex is attached to a vertex u, we mark u in order to remember this. We do not add further degree-one vertices to marked vertices. Now, we prove that for each $(v,w) \in \mathcal{L}$, the body of the for-loop in line 3 is executable in $O(|N_0^G(v,w)|)$ time. Then, the total running time of Algorithm 3.2 is $\sum_{(v,w)\in\mathcal{L}} O(|N_0^G(v,w)|)$. By Lemma 3.4, this is in $O(n)$ In the following, let G' denote the current graph, and let $(v,w) \in \mathcal{L}$ be the current pair in an iteration of the main for-loop in line 3 of Algorithm 3.2.

In the following, we show that procedure EnsurePaths can be implemented to run in $O(|N_0^G(v,w)|)$ time. Let $B(x,u)$ be defined as in procedure EnsurePaths. First, check for each vertex $u \in N_0^G(v,w)$ whether $u \in N_3^{\text{ldeg}}(x)$ in $O(1)$ time: if $B(x,u)$ only contains low-degree vertices, then $u \in N_3^{G'}(x)$ is checkable in $O(1)$ time

by simply checking whether every vertex within distance two to u in $G' - \{x\}$ is adjacent to x. If so, then $u \in N_3^{\text{ldeg}}(x)$. If, in this process, we encounter a high-degree vertex within distance two to u in $G' - \{x\}$, then clearly $B(x, u)$ does not only contain low-degree vertices and, thus, $u \notin N_3^{\text{ldeg}}(x)$. Hence, procedure EnsurePaths can compute $N_3^{\text{ldeg}}(x)$ in $O(|N_0^{G'}(v, w)|)$ time. Deleting these vertices can be done in constant time per vertex because their degrees are constant. Thus, procedure EnsurePaths runs in $O(|N_0^G(v, w)|)$ time.

Next, we show that the remainder of the main for-loop in line 3 of Algorithm 3.2 runs in $O(|N_0^G(v, w)|)$ time. The computation of N_3 in line 6 can be done by testing $u \in N_3^{G'}(v, w)$ for each $u \in N_0^G(v, w)$. By Lemma 3.3, this is possible in $O(|N_0^G(v, w)|)$ time. Testing whether N_3 can be dominated by a single vertex in $V(G') \setminus \{v, w\}$ in line 7 can be done in $O(|N_0^G(v, w)|)$ time with the following idea. First, compute the set $X := N^{G'}[N_3] \setminus \{v, w\}$, that is, N_3 and all neighbors of N_3-vertices in G' (except for v and w). Since $N_3 \subseteq N_0^G(v, w)$ and each vertex in N_3 is a low-degree vertex, X can be computed in $O(|N_0^G(v, w)|)$ time. To test the "if"-condition in line 7, it remains to check whether $N_3 \not\subseteq N^{G'}[u]$ for each $u \in X$ since vertices in N_3 can only be dominated by some vertex $u \in X$ or $\{v, w\}$. Now, by definition, all neighbors of a vertex in $N_3^{G'}(v, w)$ are in $N_0^{G'}(v, w)$, implying $X \subseteq N_0^{G'}(v, w)$. Hence, each $u \in X$ is a low-degree vertex, allowing us to test $N_3 \not\subseteq N^{G'}[u]$ in $O(1)$ time: if $|N^{G'}[u]| < |N_3|$, then, clearly, $N_3 \not\subseteq N^{G'}[u]$. Otherwise, $|N_3| \leq |N^{G'}[u]|$ and, since $N^{G'}[u]$ has constant size, we can check in $O(1)$ time whether $N_3 \not\subseteq N[u]$. Thus, the condition in line 7 can be checked in $O(|X|) \subseteq O(|N_0^G(v, w)|)$ time.

We can check whether $N_3 \subseteq N(v)$, $N_3 \subseteq N(w)$, and $N_3 \subseteq N(v, w)$ for given (v, w) in $O(|N_3|)$ time: for each vertex $u \in N_3$, simply test $v \in N(u)$ and $w \in N(u)$, respectively. Since vertices in N_3 are low-degree vertices, this test runs in $O(1)$ time for each $u \in N_3$. It follows that all conditions from lines 7 to 11 in Algorithm 3.2 can be tested in $O(|N_3|) \subseteq O(|N_0^G(v, w)|)$ time.

To execute line 8, we have to find two vertices $z, z' \in N_3$ with $\{z, z'\} \notin E(G')$ and $(N^{G'}(z) \cap N^{G'}(z')) \setminus N_3 \subseteq \{v, w\}$. By Lemma 3.2, such vertices exist. If $|N_3| \leq 78(78-1)+1$, then we can simply find z and z' by checking for each vertex pair in N_3 whether it satisfies the condition. This takes $O(1)$ time since N_3 has constant size and contains only low-degree vertices. Otherwise, $|N_3| > 78(78-1)+1$. Choose an arbitrary vertex $u \in N_3$. Observe that u is a low-degree vertex and it is (except for v and w) only adjacent to low-degree vertices. Then, since $|N_3| > 78(78-1)+1$, a vertex $z' \in N_3$ exists that is not connected to u by a path of length at most two that does not contain v and w. That is, choosing $z := u$, a vertex $z' \in N_3$ exists such that $\{z, z'\} \notin E(G')$ and $N^{G'}(z') \cap N^{G'}(z) \subseteq \{v, w\}$. We can check for each $z' \in N_3$ whether $z' \in N^{G'}(z)$ and $N^{G'}(u) \cap N^{G'}(z') \subseteq \{v, w\}$.

Since, z' is a low-degree vertex, each such check takes O(1) time. Hence, we find z and z' in $O(|N_3|) \subseteq O(|N_0^G(v, w)|)$ time.
Deletion of an N_3-vertex from G' is doable in O(1) time since only low-degree vertices are deleted. Hence, also the deletion of N_3 runs in $O(|N_0^G(v, w)|)$ time and the lemma follows. □

This concludes the presentation of our reduction rules. In the next sections, we prove that careful application of Reduction Rule 3.1 and Algorithm 3.2 allows proving a linear bound on the number of vertices in the output of our kernelization algorithm. As long as the total number of applications of these data reductions is constant, the whole kernelization algorithm runs in linear time.

3.4 Problem Kernel

In this section, we present our kernelization algorithm based on the data reduction rules shown in the previous section. We prove that, given a planar graph G, the kernelization algorithm computes a graph G' with $\gamma(G) = \gamma(G')$ whose size is linear in $\gamma(G)$ in three phases. Each phase applies Reduction Rule 3.1 or Algorithm 3.2 to finally output the problem kernel. Note that these applications are not necessarily exhaustive, but enough to bound the size of G'. The result of each phase i will be called G_i. We finally show that $|V(G_3)| \in O(\gamma(G))$.

Phase 1: Apply Reduction Rule 3.1 to each vertex of G. Let G_1 denote the resulting graph. By Lemma 3.1, G_1 is computable in $O(n)$ time and is reduced with respect to Reduction Rule 3.1.

Phase 2: Apply to G_1 (in order):

1. Algorithm 3.2
2. Reduction Rule 3.1 exhaustively
3. Algorithm 3.2
4. Reduction Rule 3.1 exhaustively

Let G_2 denote the result. By Lemmas 3.1 and 3.5, G_2 is computable in $O(n)$ time. Lemma 3.15 shows that most vertices in G_2 have constant degree.

Phase 3: Apply to G_2 (in order):

1. Algorithm 3.2
2. Reduction Rule 3.1 exhaustively

Let G_3 denote the result. Using the fact that in G_2 most vertices have only O(1) neighbors and that G_2 is reduced with respect to Reduction Rule 3.1, we will show that Phase 3 removes enough parts from G_2 to obtain the problem kernel graph G_3.

Because our kernelization algorithm applies Reduction Rule 3.1 and Algorithm 3.2 a constant number of times and since Lemmas 3.1 and 3.5 state that we can apply them in linear time, it follows that the kernelization algorithm runs in linear time. To prove that the size of G_3 is linear in $\gamma(G)$, we show that almost all vertices in G_2 have at most 78 neighbors. Exploiting this together with the fact that G_2 is reduced with respect to Reduction Rule 3.1, we can show that, in Phase 3, Algorithm 3.2 removes enough vertices from G_2 to obtain a problem kernel whose size is linear in $\gamma(G)$. To this end, let $\mathfrak{D}'$ denote the set of vertices in $V(G_1) \cap V(G_3)$ to which dummy vertices have been attached in phases two or three, that is,

$$\mathfrak{D}' := \{v \in V(G_1) \cap V(G_3) : N^{G_3}(v) \setminus V(G_1) \neq \emptyset\}.$$

Our argumentation then evolves around a dominating set $\mathfrak{D} \supseteq \mathfrak{D}'$ of size at most $2\gamma(G)$ for G_1. The "extension" $\mathfrak{D}$ of $\mathfrak{D}'$ incorporates this factor-two blowup because of one case (namely if $N_3 \subseteq N^{G'}(v) \cap N^{G'}(w)$ in line 8) of Algorithm 3.2) in which we cannot tell whether v or w is in an optimal dominating set.
We use a maximal $\mathfrak{D}$-region decomposition of G_1 with respect to an arbitrary embedding. Note that we may not be able to efficiently compute $\mathfrak{D}$. However, $\mathfrak{D}$ and the embedding of G_1 are only required for the analysis. Before using a maximal $\mathfrak{D}$-region decomposition of G_1 to prove that structure of G_1 transfers to G_2 and G_3, we show that $\mathfrak{D}$ exists.

Lemma 3.6. *There is a dominating set $\mathfrak{D} \supseteq \mathfrak{D}'$ for G_1 with $|\mathfrak{D}| \leq 2\gamma(G)$.*

Proof. We first show that dummy vertices are never attached to other dummy vertices. This implies that dummy vertices, in particular those contained in $V(G_1) \setminus V(G_3)$, always remain degree-one vertices. For the sake of contradiction assume that G' and G'' are two intermediate graphs in the computation of G_3 such that

1. G' is the the last graph in which no two dummy vertices are adjacent and
2. G'' is the first graph such that there is a dummy vertex $v \in V(G') \cap V(G'')$ that is adjacent to another dummy vertex $u \in V(G'') \setminus V(G')$.

Then, u is attached to v by some data reduction rule. Clearly, since Algorithm 3.2 does not apply to dummy vertices, u is attached to v by Reduction Rule 3.1. Then, $|N_3^{G'}(v)| > 1$ or $|N^{G'}(N_3^{G'}(v))| > 1$. Since v is not adjacent to dummy vertices in G', it follows in both cases that v has at least two non-dummy neighbors in G', contradicting the fact that dummy vertices are attached only to single vertices.
Now we construct the dominating set $\mathfrak{D}$. Each vertex $v \in \mathfrak{D}'$ has a neighbor in $u \in V(G_3) \setminus V(G_1)$, which is a degree-one vertex. Since $N[u] \subseteq N[v]$, the vertex v, and, analogously, all other vertices in $\mathfrak{D}'$ can be assumed to be part of a minimum dominating set of G_3. That is, $|\mathfrak{D}'| \in O(\gamma(G_3))$. Now, let $\mathfrak{D}^*$ be a

minimum dominating set of G_1. Obviously, also $\mathfrak{D} := \mathfrak{D}' \cup \mathfrak{D}^*$ dominates G_1. Since our data reduction rules are correct, we have $\gamma(G_1) = \gamma(G_3) = \gamma(G)$ and, therefore, $|\mathfrak{D}| \leq 2\gamma(G)$. □

In the rest of the chapter, we will base our reasoning on a fix $\mathfrak{D}$-region decomposition $\mathfrak{R}$ of G_1 with $|\mathfrak{R}| \in O(|\mathfrak{D}|) = O(\gamma(G))$. As shown by Alber et al. [8, Proposition 1], $\mathfrak{R}$ exists. Notice that, if a degree-one dummy vertex is attached to a vertex v, then neither Reduction Rule 3.1 nor Algorithm 3.2 can delete v. Instead, v is in G_3, where it still has a dummy neighbor. Then, $v \in \mathfrak{D}' \subseteq \mathfrak{D}$ and therefore, v is an anchor in the $\mathfrak{D}$-region decomposition $\mathfrak{R}$. Hence, we can observe the following.

Observation 3.3. *Let $\mathcal{R}(v, w) \in \mathfrak{R}$ be a region. Reduction Rule 3.1 and Algorithm 3.2 do not attach dummy vertices to any vertex in $\mathcal{R}(v, w) \setminus \{v, w\}$.*

Observation 3.3 ensures that, in none of the graphs G_1, G_2, and G_3, the inner vertices of a region $\mathcal{R} \in \mathfrak{R}$ have neighbors that are not in $\mathcal{R}$. In this way, Observation 3.3, to a certain extent, preserves the region structure of G_1 in G_2 and G_3. Our proof of the problem kernel size then works as follows: Proposition 3.1 shows that Phase 3 shrinks the number of vertices inside of regions of $\mathfrak{R}$ to $O(\gamma(G))$ and Proposition 3.2 shows that Phase 2 shrinks the number of vertices that are not in any region of $\mathfrak{R}$ to $O(\gamma(G))$. The proof of these main propositions are deferred to Sections 3.4.2 and 3.4.3, respectively. In the proof of Proposition 3.1, we use the essential fact that, as a result of Phase 2, inner vertices of regions in $\mathfrak{R}$ that are also present in G_2 have constant degree in G_2. This is stated as Lemma 3.7 and we dedicate Section 3.4.1 to proving it.

Lemma 3.7. *Let $\mathcal{R}(v, w)$ be a region in $\mathfrak{R}$ and let $u \in V(G_2) \setminus \{v, w\}$. If $\{v, w\} \subseteq V(G_2)$, then $|N^{G_2}(u) \cap \mathcal{R}(v, w)| \leq 78$.*

Proposition 3.1. *Let $\mathcal{R}(v, w) \in \mathfrak{R}$. Then, $|V(G_3) \cap \mathcal{R}(v, w)| \in O(1)$.*

Proposition 3.2. $|(V(G_1) \setminus V(\mathfrak{R})) \cap V(G_2)| \in O(\gamma(G))$.

Finally, we bound the number of vertices added by our kernelization algorithm (dummy vertices). Since, by Lemma 3.6, $|\mathfrak{D}'| \in O(\gamma(G))$ and since each vertex is incident to at most one dummy vertex, it follows that there are at most $O(\gamma(G))$ vertices in $V(G_3) \setminus V(G_1)$. We conclude that G_3 consists of $O(\gamma(G))$ vertices, yielding our central theorem.

Theorem 3.1. Dominating Set *on planar graphs admits a problem kernel of size $O(\gamma(G))$. This kernel can be computed in linear time.*

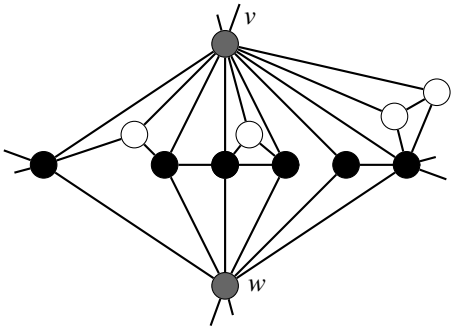


Figure 3.6: A raw diamond (A, U, P) as defined in Definition 3.5. Gray vertices v and w are the "anchors" A, black vertices are "umpires" U and white vertices are "pendants". Since the far left and right umpires have neighbors outside the raw-diamond, they are "boundary vertices" U_B. All pendants are adjacent to v and separated from $V \setminus N[v]$ by $U \cup \{v\}$.

3.4.1 The Degree of Inner Vertices of Regions (Proof of Lemma 3.7)

This section shows that if a region $\mathcal{R}(v, w)$ of the $\mathfrak{D}$-region decomposition $\mathfrak{R}$ of G_1 contains a vertex $u \notin \{v, w\}$ of high degree, then we can find a diamond-like substructure of $\mathcal{R}$, called a "raw diamond" (see Figure 3.6). We also show that such raw diamonds are shrunk to constant size by Algorithm 3.2 in Phase 2. We obtain the result that raw diamonds in G_1 have constant size in G_2 and, from this, obtain results on the maximum degree of vertices in $\mathcal{R}(v, w)$—thus proving Lemma 3.7.

Definition 3.5. *Let $G = (V, E)$ be a planar graph and let $v, w \in V$ be distinct. A* raw diamond $\mathcal{D}(v, w)$ *in G is a triple (A, U, P) such that $U, P \subseteq V$, $A \in V \times V$, and*

"anchors" $A := (v, w)$,

"umpires" $U \subseteq N(v) \cap N(w)$, *and*

"pendants" $P \subseteq N(v) \setminus (\{v, w\} \cup U)$ *such that $\{v\} \cup U$ separates P and $V \setminus N[v]$.*[23]

Let $V(\mathcal{D}(v, w))$ denote the set $\{v, w\} \cup U \cup P$, let $U_I := \{u \in U : N(u) \subseteq V(\mathcal{D}(v, w))\}$ denote the inner vertices *of $\mathcal{D}(v, w)$, and let $U_B := U \setminus U_I$ denote the* boundary vertices. *If $\deg(u) \leq 40$ for all $u \in U_I$, then $\mathcal{D}(v, w)$ is a* low-degree diamond. *For convenience, we also use A as a set, meaning $\{v, w\}$ instead of (v, w).*

[23] That is, all paths between a vertex in P and a vertex in $V \setminus N[v]$ contain a vertex in $\{v\} \cup U$.

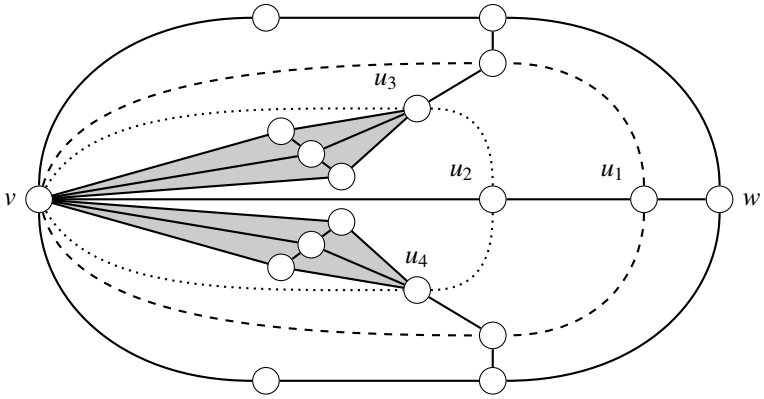


Figure 3.7: An example of a region $\mathcal{R}(v, w)$ in a planar embedding of a graph. The vertices $u_1, \ldots, u_4$ are inner vertices of the region. The gray areas form low-degree diamonds $\mathcal{D}(v, u_3)$ and $\mathcal{D}(v, u_4)$.

Also, we define the empty set of pendants to be separated from any vertex-set by any vertex-set.

We can observe that deleting non-anchors does not destroy raw-diamonds. This will come in handy to simplify some proofs.

Observation 3.4. *Let $\mathcal{D}(v, w) = (A, U, P)$ be a raw diamond in a graph G' and let G^* be a vertex-induced subgraph of G' such that $v, w \in V(G^*)$. Then, (A, U', P') with $U' := U \cap V(G^*)$ and $P' := P \cap V(G^*))$ is a raw diamond in G^* and $U'_B \subseteq U_B$.*

In the following, we explain the role of raw diamonds in shrinking regions using the example that is illustrated in Figure 3.7.

Example 1. Figure 3.7 shows a region $\mathcal{R}(v, w) \in \mathfrak{R}$ of G_1. We will see that regions whose inner vertices have constant degree are shrunk to constant size. Before that, the degrees of the inner vertices in $\mathcal{R}(v, w)$ have to be shrunk. The vertex u_1 shown in Figure 3.7 is an example for an inner vertex. Therefore, all of its neighbors are again in $\mathcal{R}(v, w)$ and, hence, in $N[v] \cup N[w]$. It follows that if u_1 has large degree, then one of $N(u_1) \cap N(v)$ or $N(u_1) \cap N(w)$ is large. Here, assume that $N(u_1) \cap N(v)$ is large. Since $N(u_1) \cap N(v) \subseteq \mathcal{R}(v, w)$, we can find a region $\mathcal{R}(v, u_1)$ in $\mathcal{R}(v, w)$ that contains $N(u_1) \cap N(v)$ and whose boundary paths have length two, like the one surrounded by the dashed line in Figure 3.7. Since any inner vertex $x \in \mathcal{R}(v, u_1)$ is in $\mathcal{R}(v, w)$, it holds that $x \in N[v]$, as, for planarity reasons, $x \notin N[w]$. One can

thus see that there is a raw diamond $\mathcal{D}(v, u_1) = (A, U, P)$ with $U = N(v) \cap N(u_1)$ and $P = \{u_3, u_4\}$ implying $V(\mathcal{D}(v, u_1) = V(\mathcal{R}(v, u_1))$.
We will show that Algorithm 3.2 shrinks raw diamonds like $\mathcal{D}(v, u_1)$ to constant size if they only contain low-degree vertices. However, suppose that u_2 is not a low-degree vertex. Since u_2 is an inner vertex of $\mathcal{R}(v, u_1)$ (the area surrounded by the dashed line), the planarity of G_1 implies $u_2 \notin N[w]$. Moreover, no inner vertex of $\mathcal{R}(v, u_1)$ is adjacent to w. Hence, all neighbors of u_2 that are not on the boundary of $\mathcal{R}(v, u_1)$, are adjacent to v and, therefore, it is sufficient to shrink $N(u_2) \cap N(v)$ in order to decrease the degree of u_2 to a constant. If $N(u_2) \cap N(v)$ is large, then, since $N(u_2) \cap N(v) \subseteq \mathcal{R}(v, u_1)$, we can find a region $\mathcal{R}(v, u_2)$ whose boundary paths have length two and $N(u_2) \cap N(v) \subseteq \mathcal{R}(v, u_2)$, which is surrounded by the dotted line in Figure 3.7. Observe that $\mathcal{R}(v, u_2)$ can be decomposed into three sets A', U', P' of a raw diamond $\mathcal{D}(v, u_2) = (A', U', P')$ with $\{u_3, u_4\} \subseteq U'$ and the vertices in the gray areas being in P'. Observe that

1. all vertices of $\mathcal{R}(v, u_1)$ are in $N[v]$,
2. all vertices of $\mathcal{R}(v, u_2)$ are in $N_{2,3}(v)$, and
3. all vertices of $\mathcal{R}(v, u_3)$ and $\mathcal{R}(v, u_4)$ are in $N_3(v)$.

Hence, $N(u_3) \cap N(v) \subseteq N_3(v)$, implying that $|N(u_3)| \in \mathrm{O}(1)$ since G_1 is reduced with respect to Reduction Rule 3.1. Then, Algorithm 3.2 shrinks $\mathcal{D}(v, u_2)$ since it only contains low-degree vertices.
We will show that for all vertices $u \in V(\mathcal{D}(v, u_1))$, Phase 2 shrinks $N(v) \cap N(u)$ to constant size (by shrinking raw diamonds like $\mathcal{D}(v, u_2)$) in the first application of Algorithm 3.2. Thereafter, Phase 2 shrinks $\mathcal{D}(v, u_1)$ in the second application of Algorithm 3.2. We will similarly show that Phase 2 also shrinks $N(v) \cap N(u)$ and $N(w) \cap N(u)$ for all other inner vertices $u \in \mathcal{R}(v, w)$. Concluding the example, Phase 3 shrinks $\mathcal{R}(v, w)$ to constant size.

The following lemma establishes properties of raw diamonds (A, U, P) in a planar graph G that are used throughout the rest of this chapter.

Lemma 3.8. *Let $\mathcal{D}(v, w) = (A, U, P)$ be a raw diamond in a planar graph G. Then, the following statements hold.*

1. *For each $x \in V(G) \setminus \{v, w\}$, it holds that $|N(x) \cap U| \leq 2$.*
2. *$N[P] \subseteq N[v]$ (and, as a direct consequence, $P \subseteq N_{2,3}(v)$).*
3. *There is a raw diamond (A, U, X) with $X \supseteq P$ and $N[X] \subseteq A \cup U \cup X$.*

Proof. (1): If there was some $x \in V(G) \setminus \{v, w\}$ such that $N(x)$ contained distinct vertices $u_1, u_2, u_3 \in U$, then G would contain a $K_{3,3}$-minor on the vertex set $\{x, v, w\} \uplus \{u_1, u_2, u_3\}$, contradicting the planarity of G.
(2): By Definition 3.5, P is separated from $V(G) \setminus N[v]$ by some vertex set, implying $N[P] \cap (V(G) \setminus N[v]) = N[P] \setminus N[v] = \emptyset$.

(3): Consider the subgraph $G' - (\{v, w\} \cup U)$ of G and the set X of all vertices that are in a connected component of G' that also contains a vertex of P. Then, $N^G[X] \subseteq \{v, w\} \cup U \cup X$ is apparent and, since $P \cap (\{v, w\} \cup U) = \emptyset$, we know that $X \supseteq P$. In the following, we show that (A, U, X) is a raw diamond. To this end, consider a vertex $x \in X$. By definition of X, there is a path p in G' from some $q \in P$ to x. Since G' does not contain vertices in $\{v\} \cup U$, this path would contradict $q \in P$ if $x \notin N^G[v]$. Since x is a vertex of G', it follows that $X \subseteq N^G[v] \setminus (\{v, w\} \cup U)$. By definition of G', it follows that $\{v, w\} \cup U$ separates X from $V \setminus N[v]$. Now, if $w \in N[X]$, then all paths from w to some vertex in $V \setminus N[v]$ contain a vertex in $\{v\} \cup U$, since otherwise, there is a path from some $q \in P$ via w to a vertex in $V \setminus N[v]$ avoiding $\{v\} \cup U$, which contradicts (A, U, P) being a raw diamond. Hence, also $\{v\} \cup U$ separates X from $V \setminus N[v]$, implying that (A, U, X) is a raw diamond. □

Since G_1 is reduced with respect to Reduction Rule 3.1, we can show additional properties of (pendants of) raw diamonds in G_1. In particular, pendants have small degree in G_1 and most pendants have a neighboring umpire.

Lemma 3.9. *Let $\mathcal{D}(v, w) = (A, U, P)$ be a raw diamond in a planar graph G that is reduced with respect to Reduction Rule 3.1. Then, for each $q \in P$, it holds that* $\deg(q) \leq 6$.

Proof. By Lemma 3.8(3), there is a raw diamond $\mathcal{D}'(v, w) = (A, U, X)$ with $X \subseteq V(\mathcal{D}'(v, w))$ and $q \in X$. By Lemma 3.8(1), $|N(q) \cap U| \leq 2$. Thus, it remains to show that $Y := N(q) \cap X \setminus \{w\}$ contains at most two vertices. By Lemma 3.8(2), $Y \subseteq N_{2,3}(v)$ and since G is reduced with respect to Reduction Rule 3.1, $Y \subseteq N_2(v)$ follows. Thus, $N[Y] \cap N_1(v) \subseteq \{w\} \cup U$ and there is a path from w (via U) to each vertex in Y avoiding both q and v. Thus, if there were three distinct vertices $y_1, y_2, y_3 \in Y$, then contracting these paths would reveal a $K_{3,3}$-minor on the vertex set $\{v, w, q\} \uplus \{y_1, y_2, y_3\}$ in G. □

Lemma 3.10. *Let $\mathcal{D}(v, w) = (A, U, P)$ be a raw diamond in a planar graph that is reduced with respect to Reduction Rule 3.1. Then, there is at most one degree-one (dummy) vertex $x \in P$ such that for all $y \in P \setminus \{x\}$, it holds that $N(y) \cap U \cap N_1(v) \neq \emptyset$.*

Proof. By Lemma 3.8, for each $q \in P$ we have $N[q] \subseteq N[v]$, and $P \subseteq N_2(v) \cup N_3(v)$. Furthermore, since the graph is reduced with respect to Reduction Rule 3.1, $N_3(v)$ is either empty or consists of a single degree-one-vertex. It follows that each vertex $q \in P$ with $\deg(q) > 1$ is in $N_2(v)$ and, hence, adjacent to at least one vertex $u \in N_1(v)$. That is, there is a vertex $u' \in N(u) \setminus N[v]$. Now, $u \in U$ follows since, by Definition 3.5, $\{v\} \cup U$ separates q from u'. Hence, $u \in U \cap N_1(v)$. □

The following lemma captures the fact that for each vertex u in a region $\mathcal{R}(v,w) \in \mathfrak{R}$, all neighbors of u are contained in one of two raw diamonds $\mathcal{D}(v,u)$ and $\mathcal{D}(v,w)$. As a result, if raw diamonds are shrunk, then the degree of u decreases as well. When the degree of all inner vertices of $\mathcal{R}(v,w)$ becomes sufficiently small, Phase 3 finds and shrinks regions to O(1) vertices.

Lemma 3.11. *Let $\mathcal{R}(v,w)$ be a region in a plane graph G. Let $u \in \mathcal{R}(v,w) \setminus \{v,w\}$, and let $a \in \{v,w\}$. Then, there is a raw diamond $\mathcal{D}(a,u) = (A,U,P)$ with $U := N(a) \cap N(u) \cap \mathcal{R}(v,w)$ and boundary size $|U_B| \leq 7$.*

Proof. Since the proof works analogously for $a = v$ and $a = w$, by exchanging the roles of v and w, we prove Lemma 3.11 only for $a = v$. Thus, let $A := \{v,u\}$ and $U := N(v) \cap N(u) \cap \mathcal{R}(v,w)$. Furthermore, we let $P \subseteq N(v) \setminus (A \cup U)$ be maximal under the condition that P is separated from $V \setminus N[v]$ by $\{v\} \cup U$. Clearly, by Definition 3.5, (A,U,P) is a raw diamond.

It remains to show that $|U_B| \leq 7$. Since $v \notin U$ and $|\partial\mathcal{R}(v,w)| \leq 6$ implies $|U_B \cap \partial\mathcal{R}(v,w)| \leq 5$, it is sufficient to show that $|U_B^*| \leq 2$, where $U_B^* := U_B \setminus \partial\mathcal{R}(v,w)$. To this end, we show that each vertex $u_1 \in U_B^*$ is connected to w by a path p such that $(V(p) \setminus \{u_1,w\}) \cap (A \cup U_B^*) = \emptyset$. Hence, if there are three distinct vertices $x_1,x_2,x_3 \in U_B^*$, then, by contracting the corresponding paths to a single edge, it follows that G contains a $K_{3,3}$ minor on the vertex set $\{v,w,u\} \uplus \{x_1,x_2,x_3\}$: this follows from the fact that $x_1,x_2,x_3 \in U_B^* \subseteq N(v) \cap N(u)$ and that each of x_1,x_2, and x_3 is connected to w by a path avoiding v,u, and U_B^*. Since the existence of this $K_{3,3}$ minor would contradict G being planar, we conclude that $|U_B^*| \leq 2$. Hence, it remains to show that each vertex $u_1 \in U_B^*$ is connected to w by a path p such that $(V(p) \setminus \{u_1,w\}) \cap (A \cup U_B^*) = \emptyset$.

Consider a vertex $u_1 \in U_B^*$. By Definition 3.5 of U_B, the vertex u_1 has a neighbor u_2 such that $u_2 \in N(u_1) \setminus (A \cup U \cup P)$. By choice of P, $u_2 \notin P$ implies that either $u_2 \notin N(v) \setminus (A \cup U)$ or u_2 is connected to a vertex in $V \setminus N[v]$ by a path that avoids $\{v\} \cup U$.

First, assume that $u_2 \notin N(v) \setminus (A \cup U)$. Since $u_1 \in U_B^*$ and $U_B^* \subseteq \mathcal{R}(v,w)$, it follows that $u_2 \in \mathcal{R}(v,w)$. Moreover, since $u_2 \notin A \cup U$, it follows that $u_2 \notin N(v)$ and thus $u_2 \in N[w]$. Hence, there is a path from u_1 via u_2 to w, where $u_2 \notin A \cup U_B^*$.

Second, assume that u_2 is connected to a vertex $t \in V \setminus N[v]$ by a path p such that $V(p) \cap (\{v\} \cup U) = \emptyset$. If $V(p) \subseteq \mathcal{R}(v,w) \setminus (A \cup U)$, then $t \in N[w]$ and there is a path from u_1 via t to w that avoids $A \cup U$ and, hence, also $A \cup U_B^*$. In the following, we assume that $V(p) \not\subseteq \mathcal{R}(v,w) \setminus (A \cup U)$. Since $u_2 \in \mathcal{R}(v,w) \setminus (A \cup U)$, it follows that there is a maximal-length subpath p' of p that contains u_2 and $V(p') \subseteq \mathcal{R}(v,w) \setminus (A \cup U)$. Let x denote the first vertex of p that is not in p' and let y denote the predecessor of x in p, that is, the last vertex on p'. Consider the following cases:

Case 1: $x \in \mathcal{R}(v,w)$. Then, by maximality of p', $x \in A \cup U$. However, since $V(p') \cap (\{v\} \cup U) = \emptyset$, this implies $x = u$. If $y \in N[v]$, then $y = v$ or $y \in N(v) \cap N(u) \cap \mathcal{R}(v,w) = U$, contradicting our choice of p. Hence, $y \in N[w]$ and appending w to p' yields a path from u_1 to w that avoids $A \cup U$ and, therefore, also $A \cup U_B^*$.
Case 2: $x \notin \mathcal{R}(v,w)$. Then, since $y \in \mathcal{R}(v,w) \cap N(x)$, we know that $y \in \partial\mathcal{R}(v,w)$. If $y \in N[w]$, then appending w to p' yields a path from u_1 to w that avoids $A \cup U$ and also $A \cup U_B^*$. Otherwise, $y \in N(v)$ since $y = v$ contradicts our choice of p. Consider the boundary path in $\partial\mathcal{R}(v,w) \setminus \{v\}$ from y to w, which has length at most two. If this boundary path contains u, then u is adjacent to y and, hence, $y \in N(u) \cap N(v) \cap \mathcal{R}(v,w) = U$, contradicting our choice of p. Otherwise, p' and the boundary path from y to w constitute a path from u_1 to w that avoids $A \cup U_B^*$. □

Lemma 3.11 guarantees the existence of a raw diamond in each region of $\mathfrak{R}$. We will show how to find and shrink these raw diamonds in order to lower the degree of vertices in regions. Using data reduction, we first ensure that most vertices in raw diamonds have constant degree. Then, we show that Algorithm 3.2 shrinks raw diamonds.
The following lemma shows that each raw diamond $\mathcal{D}(v,w) = (A,U,P)$ in G_1 with $U \neq \emptyset$ and $N(P) \subseteq \mathcal{D}(v,w)$ contains low-degree diamonds. Thereafter, we will see that these low-degree diamonds are shrunk by the first application of Algorithm 3.2 in Phase 2, effectively turning $\mathcal{D}(v,w)$ into a low-degree diamond. Thus, $\mathcal{D}(v,w)$ is shrunk by the second application of Algorithm 3.2 in Phase 2.

Lemma 3.12. *Let $\mathcal{D}(v,w) = (A,U,P)$ be a raw diamond with $N(P) \subseteq V(\mathcal{D}(v,w))$ in a planar graph that is reduced with respected to Reduction Rule 3.1. For each $u \in U$, there is a low-degree diamond $(\{v,u\}, U', \emptyset)$ with $|U'_B| \leq 7$, where $U' := N(v) \cap N(u) \cap V(\mathcal{D}(v,w))$.*

Proof. First, we show that $\mathcal{D}(v,u) := (\{v,u\}, U', \emptyset)$ is a low-degree diamond. Clearly, $\mathcal{D}(v,u)$ is a raw diamond. It remains to show that the degree of all inner vertices U'_I is at most 40 and that $|U'_B| \leq 5$. We first prove the degree-bound for vertices in U'_I. To this end, consider a vertex $u' \in U'_I$. Since u' is an inner vertex of $\mathcal{D}(v,u)$, it follows that $N(u') \subseteq V(\mathcal{D}(v,u)) = \{v,u\} \cup U'$. By Lemma 3.8(1), $|N(u') \cap U'| \leq 2$ and, thus, $|N(u')| \leq 4$. Hence, $\mathcal{D}(v,u)$ is a low-degree diamond. Next, we show that $|U'_B| \leq 7$. Observe that $U' \subseteq U \cup P \cup \{w\}$. Since $U' \subseteq N(u)$ and since, by Lemma 3.8(1), $|N(u) \cap U| \leq 2$, in order to show $|U'_B| \leq 5$ it is sufficient to show that $|U'_B \cap P| \leq 2$. Towards a contradiction, we show that for each vertex $q \in U'_B \cap P$ there is a path p from q to w such that $(V(p) \setminus \{q,w\}) \cap ((U'_B \cap P) \cup \{v,u\}) = \emptyset$. This implies $|U'_B \cap P| \leq 2$ because if there were at least three distinct vertices $q_1, q_2, q_3 \in U'_B \cap P$, then contracting the corresponding paths to an edge would reveal a $K_{3,3}$-minor on the vertex set $\{q_1,q_2,q_3\} \uplus \{v,w,u\}$, contradicting the planarity of the graph.

Let $q \in U'_B \cap P$. Then, $q \in V(\mathcal{D}(v,u)) \cap V(\mathcal{D}(v,w))$. Since $N[P] \subseteq V(\mathcal{D}(v,w))$, the definition of U'_B implies that q has a neighbor $q' \in V(\mathcal{D}(v,w)) \setminus V(\mathcal{D}(v,u))$. In case $q' \in U \cup \{w\} \subseteq N[w]$ we have found the desired path from q to w. Hence, suppose that $q' \in P$. Because q' is adjacent to both q and v, it follows that q' is not a degree-one-vertex. Since G is reduced with respect to Reduction Rule 3.1, Lemma 3.10 implies that q' is adjacent to a vertex $u' \in U$. Moreover, $u' \neq u$ since, otherwise, the choice of U' would imply $q' \in U'$, contradicting $q' \notin V(\mathcal{D}(v,u))$. Hence, we have the desired path from q via q' and u' to w. □

We can now show that Algorithm 3.2 shrinks all raw diamonds to low-degree diamonds. To this end, recall from the beginning of Section 3.4 that $\mathfrak{D}$ is a dominating set of G_1 that contains all vertices to which dummy vertices are attached by Reduction Rule 3.1 or Algorithm 3.2. We first show that, in a planar graph that is reduced with Reduction Rule 3.1, the size of raw diamonds is determined by their boundary vertices. We then apply this to G_1, showing that all umpires in raw diamonds in G_1 have small degree. Therefore all raw diamonds in G_2 are small.

Lemma 3.13. *Let G be a planar graph that is reduced with respect to Reduction Rule 3.1. Let $\mathcal{D}(v,w) = (A,U,P)$ be a low-degree diamond in G such that $V(\mathcal{D}(v,w)) \cap \mathfrak{D} \subseteq \{v,w\}$ and $N[P] \subseteq V(\mathcal{D}(v,w))$. Let G^* be the graph that is obtained by applying Algorithm 3.2 once to G and let $\{v,w\} \in V(G^*)$. Then, $|U \cap V(G^*)| \leq 5|U_B| + 3$.*

Proof. To show $|U \cap V(G^*)| \leq 5|U_B| + 3$, we prove $|U_I \cap V(G^*)| \leq 4|U_B| + 3$. Since $\mathcal{D}(v,w)$ is a low-degree diamond, each vertex $u \in U_I$ lies on a low-degree path (v,u,w) in $N^G[v,w]$. Therefore, $U_I \subseteq N_0^G(v,w)$. We show the claim in two parts.
Part (1): $|N_3^G(v,w) \cap U_I \cap V(G^*)| \leq 3$
Part (2): $|N_{1,2}^G(v,w) \cap U_I| \leq 4|U_B|$

(1): Obviously, $|N_3^G(v,w) \cap U_I \cap V(G^*)| \leq 1$ holds if at least one of v and w is a dummy vertex, since dummy vertices have degree one. Hence, in the following, assume that neither v nor w is a dummy vertex. Moreover, since $\{v,w\} \subseteq V(G^*)$, v and w are not deleted by the application of Algorithm 3.2. It follows that Algorithm 3.2 applied to the input graph G considers the pair (v,w) in the for-loop in line 3. Let G' denote the current graph at that point in the algorithm. Throughout this proof we make use of the observation that, for all $t \in (U \cup P) \cap V(G')$,

$$N^{G'}(t) = N^G(t) \cap V(G') \tag{3.1}$$

holds. This is because $V(\mathcal{D}(v,w)) \cap \mathfrak{D} \subseteq \{v,w\}$, that is, neither Reduction Rule 3.1 nor Algorithm 3.2 attaches dummy vertices to t, and since Algorithm 3.2 does not delete edges without deleting their endpoints.

Consider the set $N_3 = N_3^{G'}(v,w) \cap N_0^G(v,w)$, which is the set of candidates for deletion that is computed in line 6 of Algorithm 3.2. In the following, we show that

$$X := N_3^G(v,w) \cap U_I \cap V(G') \subseteq N_3.$$

To this end, consider a vertex $u \in X$. Since $u \in N_0^G(v,w)$, it is sufficient to show $u \in N_3^{G'}(v,w)$. First, by (3.1), u is a low-degree vertex in G' and, since $\{v,w\} \in G^*$, we know that $u \in N^{G'}(v) \cap N^{G'}(w)$. Hence, $u \in N_0^{G'}(v,w)$. To prove $u \in N_3$ it remains to show $u \in N_3^{G'}(v,w)$, or, equivalently, $u \notin N_{1,2}^{G'}(v,w)$.

- $u \notin N_1^{G'}(v,w)$: By (3.1), all vertices in $N^{G'}(u) \setminus A \subseteq U \cup P$ are low-degree vertices and since for each of them, there is a low-degree v-w-path of length at most three via u, it follows that $N^{G'}(u) \setminus A \subseteq N_0^{G'}(v,w)$. Hence, $u \notin N_1^{G'}(v,w)$.
- $u \notin N_2^{G'}(v,w)$: Suppose this is false. Then, there is a neighbor $u' \in N^{G'}(u) \cap N_1^{G'}(v,w)$ and, by Definition 3.4, u' has a neighbor $t \notin N_0^{G'}[v,w]$. However, $u \in N_3^G(v,w)$ implies $t \in N_0^G(v,w)$ and, therefore, t is a low-degree vertex and $t \in N^G(v) \cup N^G(w)$. Since $v,w,t \in V(G')$, (3.1) implies that $t \in N^{G'}(v) \cup N^{G'}(w)$. Thus, either (v,t,u',u,w) or (v,u,u',t,w) is a low-degree v-w path in G', contradicting $t \notin N_0^{G'}[v,w]$.

Now, if $|N_3| > 3$, then Lemma 3.8(1) implies that only v and w can dominate the (at least four) vertices in N_3. Thus, N_3 can be reduced by Algorithm 3.2, implying $|N_3^G(v,w) \cap U_I \cap V(G^*)| \leq 3$.

(2): It remains to show $|N_{1,2}^G(v,w) \cap U_I| \leq 4|U_B|$. To see this, we first prove

$$N_1^G(v,w) \cap (U_I \cup P) \subseteq N^G(U_B). \tag{3.2}$$

Consider a vertex $u \in N_1^G(v,w) \cap U_I$. By Definition 3.4 there is some $u' \in N^G(u) \setminus N_0^G[v,w]$. Since, by Lemma 3.9 each vertex in P has low degree and therefore $P \cap N[U_I] \subseteq N_0^G(v,w)$, we conclude $u' \notin P$. Hence, $u' \in U_B$, implying $u \in N(U_B)$ and it follows that $N_1^G(v,w) \cap U_I \subseteq N^G(U_B)$.
To prove (3.2), it remains to show $N_1^G(v,w) \cap P \subseteq N^G(U_B)$. To this end, consider a vertex $p \in N_1^G(v,w) \cap P$. By Lemma 3.10 there is a vertex $u \in N^G(p) \cap U$. If $u \in U_B$, then $p \in N^G(U_B)$. Otherwise, $u \in U_I$ and, hence, u is a low-degree vertex. By Definition 3.4 there is a vertex $t \in N^G(p) \setminus N_0^G[v,w]$. Because $N^G(p) \subseteq V(\mathcal{D}(v,w))$, we conclude $t \in U \cup P$. If $t \in U$, then $t \in U_B$ since $t \notin N_0^G[v,w]$ and $U_I \subseteq N_0^G(v,w)$. It follows that $p \in N^G(U_B)$. If $t \in P$, then t is a low-degree vertex and (v,t,p,u,w) is a low-degree v-w-path of length four in $N^G[v,w]$, implying $t \in N_0^G(v,w)$, contradicting our choice of t. Thus, (3.2) follows.

Using (3.2), we can now show $|N^G_{1,2}(v,w) \cap U_I| \leq 4|U_B|$, concluding the lemma. To this end, note that

$$N^G_{1,2}(v,w) \cap U_I \overset{\text{Definition 3.5}}{\subseteq} N^G[N^G_1(v,w) \cap (U \cup P)] \overset{(3.2)}{\subseteq} N^G[N^G(U_B)]. \tag{3.3}$$

Now, for the sake of contradiction, suppose that, for a vertex $u \in U_B$, there is a set of five distinct vertices $X \subseteq N^G_{1,2}(v,w) \cap U_I \cap N^G[N^G(u)]$. Then, we can choose at least three vertices $x_1, x_2, x_3 \in X$ such that contracting the corresponding paths reveals a $K_{3,3}$-minor on the vertex set $\{v,w,u\} \uplus \{x_1,x_2,x_3\}$. As this contradicts the planarity of G, we conclude that

$$|N^G_{1,2}(v,w) \cap U_I| \overset{(3.3)}{\leq} \sum_{u \in U_B} |N^G_{1,2}(v,w) \cap U_I \cap N^G[N^G(u)]| \leq 4|U_B|. \qquad \square$$

Approaching our goal of bounding the maximum degree of vertices in regions, we show that all raw diamonds contained in G_1 are destroyed in Phase 2, that is, by two interleaved applications of Reduction Rule 3.1 and Algorithm 3.2.

Lemma 3.14. *Let $\mathcal{D}(v,w) = (A,U,P)$ be a raw diamond in G_1 with $N^{G_1}[P] \subseteq V(\mathcal{D}(v,w))$ and $V(\mathcal{D}(v,w)) \cap \mathfrak{D} \subseteq \{v,w\}$. Furthermore, let $\{v,w\} \subseteq V(G_2)$. Then, $|U \cap V(G_2)| \leq 5|U_B| + 3$.*

Proof. Let G^* denote the graph obtained by applying Algorithm 3.2 and then Reduction Rule 3.1 to G_1. We argue that there is a low-degree diamond $\mathcal{D}'(v,w) = (A,U',P')$ in G^* such that $U' = U \cap V(G^*)$, $U'_B \subseteq U_B$, and $P' = P \cap V(G^*)$. Then, Lemma 3.14 follows directly from Lemma 3.13, since G_2 results from applying Algorithm 3.2 and additionally Reduction Rule 3.1 to G^*.
First, since our data reduction rules do not delete edges without deleting one of their endpoints, we infer

$$N^{G_1}(v) \cap V(G^*) \subseteq N^{G^*}(v) \text{ and } N^{G_1}(w) \cap V(G^*) \subseteq N^{G^*}(w). \tag{3.4}$$

Hence,

$$U' = U \cap V(G^*) \subseteq N^{G_1}(v) \cap N^{G_1}(w) \cap V(G^*) \overset{(3.4)}{\subseteq} N^{G^*}(v) \cap N^{G^*}(w), \text{ and}$$
$$P' = P \cap V(G^*) \subseteq N^{G_1}(v) \cap V(G^*) \setminus (A \cup U) \overset{(3.4)}{\subseteq} N^{G^*}(v) \setminus (A \cup U')$$

Observe that, since $V(\mathcal{D}(v,w)) \cap \mathfrak{D} \subseteq \{v,w\}$, it holds that

$$\bigvee_{u \in V(\mathcal{D}(v,w))} N^{G^*}(u) \subseteq N^{G_1}(u). \tag{3.5}$$

Since removing the vertices in $V(G_1) \setminus V(G^*)$ from G_1 does not create paths between P' and $V(G^*) \setminus N^{G^*}(v)$, (3.5) implies that P' is separated from $V(G^*) \setminus N^{G^*}(v)$ by $\{v\} \cup U'$ in G^*, implying that (A, U', P') is a raw diamond in G^*. Also note that (3.5) implies $U'_B \subseteq U_B$.
It remains to show that $\deg_{G^*}(u) \leq 40$ for each $u \in U'_I$. By Lemma 3.12 there is a low-degree diamond $\mathcal{D}(v,u) = (\tilde{A}, \tilde{U}, \emptyset)$ in G_1 with $\tilde{U} = N^{G_1}(v) \cap N^{G_1}(u) \cap V(\mathcal{D}(v,w))$ and $|\tilde{U}_B| \leq 7$. Since $u \in U'_I \subseteq U_I$, we know that $N^{G_1}(u) \subseteq V(\mathcal{D}(v,w))$ and, thus, $\tilde{U} = N^{G_1}(v) \cap N^{G_1}(u)$. By Lemma 3.13, $|\tilde{U} \cap V(G^*)| \leq 38$. Since $N^{G_1}(u) \setminus N^{G_1}(v) \subseteq \{v,w\}$, we conclude that

$$\deg_{G^*}(u) \overset{(3.5)}{\leq} \deg_{G_1}(u) \leq 38 + 2 = 40. \qquad \square$$

Finally, for each region $\mathcal{R} := \mathcal{R}(v,w)$ in the $\mathfrak{D}$-region decomposition $\mathfrak{R}$ of G_1 and each vertex $u \in \mathcal{R} \setminus \{v,w\}$ in G_2, we are able to upper-bound the number of neighbors of u in $\mathcal{R}$. If u is an *inner* vertex of $\mathcal{R}$, then this yields an upper bound on the degree of u. We will show in Section 3.4.2 that this allows us to find and shrink regions using Algorithm 3.2 in Phase 3. We state this fact as a generalization of Lemma 3.7.

Lemma 3.15. *Let $\mathcal{R} := \mathcal{R}(v,w)$ be a region in $\mathfrak{R}$, and let $u \in V(G_2) \setminus \{v,w\}$. Then,*
(i) if $v \in V(G_2)$, then $|N^{G_2}(u) \cap \mathcal{R} \cap N^{G_2}(v)| \leq 38$, and
(ii) if $w \in V(G_2)$, then $|N^{G_2}(u) \cap \mathcal{R} \cap N^{G_2}(w)| \leq 38$.
That is, if $\{v,w\} \subseteq V(G_2)$, then $|N^{G_2}(u) \cap \mathcal{R}| \leq 78$.

Proof. Because (i) and (ii) are symmetrical, we only show (i). If $u \notin R$, then $N^{G_2}(u) \cap R \subseteq \partial R$. Hence, $|N^{G_2}(u) \cap R \cap N^{G_2}(v)| \leq 6$. In the following, assume that $u \in R$. Then, by Lemma 3.11, there is a raw diamond $\mathcal{D}(v,u) = (A, U, P)$ in G_1 such that $U = N^{G_1}(u) \cap N^{G_1}(v) \cap R$ and $|U_B| \leq 7$. Because $\{v,u\} \subseteq V(G_2)$, Lemma 3.14 together with Observation 3.3 implies $|U \cap V(G_2)| \leq 38$. Moreover, because Algorithm 3.2 does not add edges between already existing vertices, $N^{G_1}(u) \cap V(G_2) \cap R = N^{G_2}(u) \cap R$.
Now, assume that $\{v,w\} \subseteq V(G_2)$. Since $u \in R \setminus \{v,w\}$, it holds that $N^{G_2}(u) \cap R \subseteq N^{G_2}[v,w]$. The claim then follows from $|N^{G_2}(u) \cap N^{G_1}(v) \cap R| \leq 38$ and $|N^{G_2}(u) \cap N^{G_1}(w) \cap R| \leq 38$. $\square$

With Lemma 3.15, we know that the degree of each inner vertex of any region is constant in G_2. This enables Algorithm 3.2 to shrink the regions to constant size.

3.4.2 The Size of Regions (Proof of Proposition 3.1)

In this section, we show that the $O(\gamma(G))$ regions in the maximal $\mathfrak{D}$-region decomposition $\mathfrak{R}$ of G_1 can be shrunk to constant size in Phase 3. A central

problem in bounding the size of a region is bounding its *core*: intuitively, the core of a region R contains all vertices of R that have long distance to R's boundary.

Definition 3.6. *Let* $\mathcal{R} := \mathcal{R}(v, w) \in \mathfrak{R}$. *The* core $\mathfrak{CR}$ *of* $\mathcal{R}$ *is* $\mathfrak{CR} := \{u \in (\mathcal{R} \setminus \partial\mathcal{R}) \cap V(G_2) : \text{dist}_{G_1[\mathcal{R}]}(u, y) \geq 5$ *for all* $y \in \partial\mathcal{R} \cap V(G_2) \setminus \{v, w\}\}$.

In order to show that regions are shrunk to constant size in Phase 3, we first show that only a constant number of vertices outside of a region's core remain after Phase 2.

Lemma 3.16. *Let* $\mathcal{R} := \mathcal{R}(v, w) \in \mathfrak{R}$ *and, for each* $u \in V(G_2)$, *let* $|N^{G_2}(u) \cap \mathcal{R}| \in$ O(1). *Then,* $|V(G_2) \cap (\mathcal{R} \setminus \mathfrak{CR})| \in$ O(1).

Proof. By definition of $\mathfrak{CR}$, each vertex $x \in V(G_2) \cap (\mathcal{R} \setminus \mathfrak{CR})$ satisfies $\text{dist}^{G_2[\mathcal{R}]}(x, y) < 5$ for a vertex $y \in \partial\mathcal{R} \setminus \{v, w\} \cap V(G_2)$. Since $|\partial\mathcal{R} \setminus \{v, w\} \cap V(G_2)| \leq 4$ and since, by assumption, each vertex $u \in V(G_2)$ satisfies $|N^{G_2}(u) \cap \mathcal{R}| \in$ O(1), there are at most O(1) paths of length at most five in $\mathcal{R} \cap V(G_2)$ that start in a vertex in $\partial\mathcal{R} \setminus \{v, w\} \cap V(G_2)$ and end in a vertex in $V(G_2) \cap (\mathcal{R} \setminus \mathfrak{CR})$. Since, by Lemma 3.15, all vertices on these paths have constant degree, $|V(G_2) \cap (\mathcal{R} \setminus \mathfrak{CR})| \in$ O(1). □

Another problem that complicates the proof of Proposition 3.1 is the fact that anchors of regions in $\mathfrak{R}$ might be deleted by our data reduction. Therefore, we will first establish that Proposition 3.1 holds in these cases using Lemma 3.18. In preparation, we prove that the set of vertices that were neighbors of a deleted anchor of $\mathcal{R}(v, w)$ is small in G_3.

Lemma 3.17. *Let* $R := \mathcal{R}(v, w) \in \mathfrak{R}$ *and* $x \in \{v, w\} \setminus V(G_3)$. *Let* G^* *be the last graph containing* x *in the computation of* G_3. *Then,* $|N^{G^*}(x) \cap R \cap V(G_3)| \in$ O(1).

Proof. We show the claim for $x = v$. If $x = w$, the proof is completely analogous. We distinguish whether Algorithm 3.2 or Reduction Rule 3.1 deletes v.
Case 1: Algorithm 3.2 deletes v. Since Algorithm 3.2 only deletes v if v is a low-degree vertex, it follows that $|N^{G^*}(v) \cap R \cap V(G_3)| \in$ O(1).
Case 2: v is deleted by applying Reduction Rule 3.1 to a vertex $v' \in N^{G^*}(v)$. Then, $v \in N_3^{G^*}(v')$, implying $N^{G^*}[v] \subseteq N^{G^*}[v']$.
First, assume that $v' \neq w$ and, furthermore, assume that there is a vertex $u \in N^{G^*}(v) \cap (R \setminus \partial R)$. Thus, $u \in N^{G^*}[v'] \cap R \setminus \partial R$ and Observation 3.3 in the beginning of Section 3.4, we conclude that $v' \in R$. Then, however, the fact that Reduction Rule 3.1 attaches a dummy vertex to v' contradicts Observation 3.3. Therefore, $N^{G^*}(v) \cap (R \setminus \partial R) = \emptyset$. Hence, we conclude $N^{G^*}(v) \cap R \subseteq \partial R$, implying $|N^{G^*}(v) \cap R| \in$ O(1).
Second, assume $v' = w$, that is, v is deleted by applying Reduction Rule 3.1 to w. Then, $N^{G^*}[v] \subseteq N^{G^*}[w]$. Furthermore, for each $u \in V(G^*) \setminus \{v, w\}$, it

holds that $|N^{G^*}(u) \cap N^{G^*}(v)| \leq 2$, since otherwise, $N^{G^*}[v] \subseteq N^{G^*}[w]$ implies the existence of a $K_{3,3}$ subgraph in G^*. Moreover, the application of Reduction Rule 3.1 to w implies $|N_3^{G^*}(w) \cap V(G_3) \cap R| \leq 1$. It remains to bound the number of vertices $u \in N^{G^*}(v) \cap R \cap V(G_3) \cap N_{1,2}^{G^*}(w)$. Since $u \notin N_3^{G^*}(w)$, there is a vertex $u' \neq u$ in $N^{G^*}[N^{G^*}[u] \setminus \{w\}] \setminus N^{G^*}[w]$. Since $N^{G^*}[w] \supseteq N^{G^*}[v]$, it follows that $u' \notin N^{G^*}[v]$ and, therefore, $u' \notin R$. Since $u \in R$ is adjacent to u' or has a common neighbor with u', we obtain $u \in N^{G^*}[\partial R]$. Since $|\partial R| \leq 6$ and for each $u \in V(G^*) \cap \partial R \setminus \{v, w\}$, it holds that $|N^{G^*}(u) \cap N^{G^*}(v)| \leq 2$, we conclude that $|N^{G^*}(v) \cap R \cap V(G_3)| \in \mathrm{O}(1)$. □

Lemma 3.18. *Let $\mathcal{R} := \mathcal{R}(v, w) \in \mathfrak{R}$ and $\{v, w\} \nsubseteq V(G_3)$. Then, $|\mathcal{R} \cap V(G_3)| \in \mathrm{O}(1)$.*

Proof. We show the claim for $v \notin V(G_3)$. If $w \notin V(G_3)$, the proof is completely analogous. In the computation of G_3 from G_1, let G^* be the last graph containing v. For $v \notin V(G_3)$, Lemma 3.17 implies

$$|N^{G^*}(v) \cap R \cap V(G_3)| \in \mathrm{O}(1). \tag{3.6}$$

If also $w \notin V(G_3)$, then $|N^{G^*}(w) \cap R \cap V(G_3)| \in \mathrm{O}(1)$, which would already yield $|R \cap V(G_3)| \in \mathrm{O}(1)$. Hence, in the following, assume that $w \in V(G_3)$, implying $w \in V(G_2) \cap V(G^*)$.
Consider a vertex $u \in V(G_3) \setminus \{v, w\}$. Towards the application of Lemma 3.16, we show

$$|N^{G_3}(u) \cap R| \in \mathrm{O}(1). \tag{3.7}$$

If $u \notin V(G_2)$, then Observation 3.3 implies $N^{G_3}(u) \cap R \subseteq \{v, w\}$ and, thus, (3.7) holds. Hence, assume $u \in V(G_2)$. Note that, if $u \notin R$, then $N^{G_3}(u) \cap R \subseteq \partial R$, implying (3.7). Hence, assume $u \in R$. Then, $N^{G_3}(u) \cap R \subseteq (N^{G_3}(v) \cup N^{G_3}(w)) \cap R$. Since $w \in V(G_2)$, Lemma 3.15 implies $|N^{G_2}(u) \cap R \cap N^{G_2}(w)| \in \mathrm{O}(1)$. Then,

$$|N^{G_3}(u) \cap R| \leq |N^{G^*}(v) \cap V(G_3) \cap R| + |N^{G_3}(u) \cap N^{G_3}(w) \cap R| \overset{(3.6)}{\in} \mathrm{O}(1).$$

Therefore, Lemma 3.16 applies and yields $|R \cap V(G_2) \setminus \mathfrak{C}R| \in \mathrm{O}(1)$, implying $|V(G_3) \cap (R \setminus \mathfrak{C}R)| \in \mathrm{O}(1)$.
It remains to show $|\mathfrak{C}R \cap V(G_3)| \in \mathrm{O}(1)$. To this end, we prove that each vertex in $\mathfrak{C}R \cap V(G_3)$ has a path of length at most two in $R \cap V(G_3)$ to a vertex in

$$U := N^{G^*}(v) \cap R \cap V(G_3).$$

By Lemma 3.17, $|U| \in \mathrm{O}(1)$ and, since (3.7) holds for all $u \in V(G_3) \setminus \{v, w\}$, there are at most $\mathrm{O}(1)$ such paths, implying $|\mathfrak{C}R \cap V(G_3)| \in \mathrm{O}(1)$. Note that it is

sufficient to bound the size of $\mathfrak{C}R \cap V(G_3) \setminus (U \cup N_3^{G_3}(w))$, since $|U| \in \mathrm{O}(1)$ and, by reducedness of G_3 with respect to Reduction Rule 3.1, $|N_3^{G_3}(w)| \leq 1$. To this end, consider a vertex

$$u \in \mathfrak{C}R \cap V(G_3) \setminus (U \cup N_3^{G_3}(w)) \subseteq \mathfrak{C}R \cap V(G_3) \setminus (N^{G^*}(v) \cup N_3^{G_3}(w)). \quad (3.8)$$

We show that there is a path of length at most two in $R \cap V(G_3)$ from u to a vertex in U. Since, by (3.8), $u \in R \setminus N^{G^*}(v)$, we conclude $u \in N^{G_3}[w]$. However, by (3.8), $u \notin N_3^{G_3}(w)$, implying the existence of a vertex $u' \in N^{G_3}[u] \cap N^{G_3}(w)$ and a vertex $u'' \in N^{G_3}(u') \setminus N^{G_3}[w]$ (possibly, $u = u'$). Observation 3.3 and $u \in \mathfrak{C}R \cap V(G_3)$ now imply $\{u', u''\} \subseteq R \cap V(G_3)$ and, since $u'' \notin N^{G_3}[w]$, we obtain $u'' \in N^{G^*}[v] \cap V(G_3) \subseteq U$. Hence, (u, u', u'') is a path of length two in $R \cap V(G_3)$ from u to a vertex in U. □

Lemma 3.18 allows us to focus on regions whose anchors survive the data reduction and exist in G_3. The strategy to prove Proposition 3.1 is to show that Algorithm 3.2 shrinks regions in Phase 3. To this end, we have to analyze Algorithm 3.2 (see page 81).

Proposition 3.1. *Let $\mathcal{R}(v, w) \in \mathfrak{R}$. Then, $|V(G_3) \cap \mathcal{R}(v, w)| \in \mathrm{O}(1)$.*

Proof. Let $\mathcal{R} := \mathcal{R}(v, w)$. If $\{v, w\} \not\subseteq V(G_3)$, then $|\mathcal{R} \cap V(G_2)| \in \mathrm{O}(1)$ by Lemma 3.18. Hence, assume that $\{v, w\} \subseteq V(G_3)$, that is, v and w are never deleted in the computation of G_3. Moreover, by choice of the maximal $\mathfrak{D}$-region decomposition $\mathfrak{R}$ for G_1, neither v nor w is a dummy vertex. We now show that $|\mathfrak{C}\mathcal{R} \cap V(G_3)| \in \mathrm{O}(1)$, which is sufficient since, by Lemma 3.15 and Lemma 3.16, $|V(G_2) \cap (\mathcal{R} \setminus \mathfrak{C}\mathcal{R})| \in \mathrm{O}(1)$, implying also $|V(G_3) \cap (\mathcal{R} \setminus \mathfrak{C}\mathcal{R})| \in \mathrm{O}(1)$.
When applied to G_2, Algorithm 3.2 eventually considers the pair (v, w) in an iteration of the for-loop in line 3. Let G' be the graph in this iteration after procedure EnsurePaths has been applied in lines 4 and 5. Since G_3 is reduced with respect to Reduction Rule 3.1, the set $Y := \{u \in \mathfrak{C}\mathcal{R} \cap V(G') : \deg_{G'}(u) = 1\}$ satisfies $|Y \cap V(G_3)| \leq 2$ because $Y \subseteq N^{G'}(v, w)$. It remains to bound the number of vertices in $\mathfrak{C}\mathcal{R} \cap V(G_3) \setminus Y$. To this end, we now show that $\mathfrak{C}\mathcal{R} \cap V(G') \setminus Y \subseteq N_3$, where $N_3 = N_3^{G'}(v, w) \cap N_0^{G_2}(v, w)$ is the set of deletion candidates computed in line 6. Note that, since our data reduction rules do not add or delete edges between existing vertices, $N_3^{G'}(v, w) \subseteq N_0^{G_2}(v, w)$ and, therefore, $N_3^{G'}(v, w) \subseteq N_3$. Hence, to show $\mathfrak{C}\mathcal{R} \cap V(G') \setminus Y \subseteq N_3$, it is sufficient to show

$$\mathfrak{C}\mathcal{R} \cap V(G') \setminus Y \subseteq N_3^{G'}(v, w). \quad (3.9)$$

To this end, let $u \in \mathfrak{C}\mathcal{R} \cap V(G')$. To prove (3.9), we show that vertices of $\mathcal{R}$ with distance at most two from u are in $N_0^{G'}(v, w)$. More formally, we show

$$X \subseteq N_0^{G'}(v, w) \text{ with } X := N^{G'}[N^{G'}[u] \setminus \{v, w\}] \setminus \{v, w\}. \quad (3.10)$$

Observe that $X \subseteq \mathcal{R}$ and, by Observation 3.3, $X \subseteq V(G_2)$. Since $u \in \mathfrak{C}\mathcal{R}$, all vertices $x \in X$ have distance at least three to the boundary $\partial\mathcal{R}$ in G_2. More formally,

$$\text{dist}_{G_2[\mathcal{R}]}(x, y) \geq 3 \text{ for all } y \in \partial\mathcal{R} \cap V(G_2) \setminus \{v, w\}. \tag{3.11}$$

It follows that $X \subseteq (\mathcal{R} \setminus \partial\mathcal{R}) \cap V(G')$ and, therefore, $X \subseteq N^{G'}(v, w)$. Moreover, by Lemma 3.15, X contains only low-degree vertices of G_2. Since our data reduction rules do not increase the degree of vertices, X also contains only low-degree vertices of G'. In the following, let $x \in X$. To prove (3.10), we show that $x \in N_0^{G'}(v, w)$. Since $x \in N^{G'}(v, w)$, without loss of generality, assume that $x \in N^{G'}(v)$. The case $x \in N^{G'}(w)$ works analogously. We consider the relation between x and v in three cases.

Case 1: $x \in N_{1,2}^{G'}(v)$. Then, there is a path p' of length at most two from x to some $x' \notin N^{G'}[v]$ and p' avoids v. If w is not on p then, since $x' \in \mathcal{R} \setminus N^{G'}[v]$, we know that w is a neighbor of x'. Hence, there is also a path p of length at most three from x to w. By (3.11), all vertices on p between x and w are in $\mathcal{R} \setminus \partial\mathcal{R}$. Thus, by Lemma 3.15, p is a low-degree-path, implying that prepending v to p yields a low-degree v-w-path containing x in G'. Therefore, $x \in N_0^{G'}(v, w)$.

Note that $x \in N_3^{G_2}(v)$ would imply $\deg_{G_2}(x) \leq 1$ since G_2 is reduced with respect to Reduction Rule 3.1. Thus, we can observe that $x \in N_0^{G_2}(v, w)$ since Case 1 also applies to $G' = G_2$. Hence, $\deg_{G'}(x) \leq 1$ and, therefore, x is only adjacent to v in G', which by choice of $x \in X$ implies $x = u$, contradicting $u \notin Y$.

Case 2: $x \in N_3^{G'}(v)$. If $\deg_{G'}(x) \leq 1$, then x is only adjacent to v, which implies $x = u$ and contradicts $u \notin Y$. Hence, $\deg_{G'}(x) > 1$. If $x \in N^{G'}(w)$, then, clearly, $x \in N_0^{G'}(v, w)$. Now, consider $x \notin N^{G'}(w)$, and let G^* denote the graph that is G' prior to the application of procedure EnsurePaths to v and w. Observe that, as discussed in the proof of Lemma 3.1, $x \in N_3^{G'}(v)$ implies $x \in N_3^{G^*}(v)$. Since procedure EnsurePaths does not delete x even though $x \in N_3^{G^*}(v) \cap N_0^{G_2}(v, w)$ and $\deg_{G^*}(x) > 1$, there is a vertex $z \in B(v, x)$ of procedure EnsurePaths that is not a low-degree vertex in G^*. By Lemma 3.15, all vertices in $V(G^*) \cap \mathcal{R} \setminus (\partial\mathcal{R} \cup \{v, w\})$ are low-degree vertices and, therefore, $z \notin \mathcal{R} \setminus (\partial\mathcal{R} \cup \{v, w\})$.

Since distances between vertices only increase while computing G^* from G_2, we infer that $\text{dist}_{G^*}(x, w) \geq 2$. Furthermore, by (3.11), $\text{dist}_{G^*[\mathcal{R}]}(x, y) \geq 3$ for all $y \in \partial\mathcal{R} \cap V(G^*) \setminus \{v, w\}$. Since $z \notin \mathcal{R} \setminus (\partial\mathcal{R} \cup \{v, w\})$, this implies $z = w$ and, thus, $w \in B(v, x)$. Hence, there is a length-two low-degree path p from x to w in G^*. We now argue that there is a low-degree path p' from x to w in G'. If p is not a low-degree path in G', then procedure EnsurePaths deleted a vertex y of p.

If y was deleted by the application of procedure EnsurePaths to v, then, by definition of $B(v, x)$, we know that $y \in N^{G^*}(x) \cap N^{G^*}(w)$. Then, however, $w \in B(v, y)$, which is not a low-degree vertex, and procedure EnsurePaths does not delete y.

If y is deleted by the application of procedure EnsurePaths to w, then $y \in N_3^{G^*}(w)$ implies that $x \in N^{G^*}(w)$. Thus, $x \in N^{G'}(w)$, contradicting $x \notin N^{G'}(w)$.

In all cases, we conclude that $x \in N_0^{G'}(v, w)$. Since $x \in X$ was chosen arbitrarily, (3.10) follows and, as discussed in the beginning of the proof, $\mathfrak{CR} \cap V(G') \setminus Y \subseteq N_3$. We finally show that Algorithm 3.2 shrinks N_3 to O(1) vertices. First, assume that N_3 can be dominated by a single vertex $u' \in V(G') \setminus \{v, w\}$. Since u' is a neighbor of a vertex in $N_3^{G'}(v, w)$, u' is a low-degree vertex and we obtain $|N_3| \leq |N^{G'}[u']| \in \mathrm{O}(1)$. If, in contrast, N_3 cannot be dominated by a single vertex $u' \in V(G') \setminus \{v, w\}$, then N_3 is deleted except for at most two vertices; we obtain $|\mathfrak{CR} \cap V(G_3) \setminus Y| \in \mathrm{O}(1)$. □

3.4.3 Vertices Outside of Regions (Proof of Proposition 3.2)

In this section, we show that only $\mathrm{O}(\gamma(G))$ vertices outside of regions in the maximal $\mathfrak{D}$-region decomposition $\mathfrak{R}$ of G_1 remain after Phase 2. In particular, we show slightly modified versions of Lemma 6, Lemma 7, and Proposition 2 of Alber et al. [8]. To this end, we exploit the structure that Phase 1 establishes in G_1. First, the following lemma implies that no vertex outside of regions is contained in $N_1^{G_1}(v)$ for a vertex $v \in \mathfrak{D}$. It can be shown analogously[24] to Lemma 6 of Alber et al. [8] and, hence, we omit the proof.

Lemma 3.19. *Let $\mathfrak{D}$ be a dominating set of G_1 and let $v \in \mathfrak{D}$ and $u \in N_1^{G_1}(v)$. Let $\mathfrak{R}$ denote a $\mathfrak{D}$-region decomposition such that $|\mathfrak{R}| \in \mathrm{O}(|\mathfrak{D}|)$. Then, $u \in V(\mathfrak{R})$.*

Using Lemma 3.19, we can show that vertices that are not in regions of $\mathfrak{R}$ or in $\mathfrak{D}$ are in $N_2^{G_1}(v)$ for some $v \in \mathfrak{D}$, which we use later to bound the total number of vertices that are not in regions of $\mathfrak{R}$.

Lemma 3.20. *Let $u \in V(G_1) \setminus (\mathfrak{D} \cup V(\mathfrak{R}))$ with $\deg_{G_1}(u) > 1$. Then, there is a vertex $v \in \mathfrak{D}$ such that $u \in N_2^{G_1}(v)$.*

Proof. Consider a vertex $u \in V(G_1) \setminus (\mathfrak{D} \cup V(\mathfrak{R}))$. Since $\mathfrak{D}$ is a dominating set of G_1, there is some vertex $v \in N^{G_1}(u) \cap \mathfrak{D}$. By Lemma 3.19, we know that $u \notin N_1^{G_1}(v)$. Furthermore, since G_1 is reduced with respect to Reduction Rule 3.1, either $u \notin N_3^{G_1}(v)$ or u is a degree-one vertex. Since, by assumption, u is not a degree-one vertex, the claim follows. □

[24] Although Alber et al. [8] required reducedness of G_1 with respect to *their* reduction rules, the two statements we use are also true for *our* reduction rules. In fact, the proof of Alber et al. [8] does not require reducedness at all and is, instead, only based on the maximality of the $\mathfrak{D}$-region decomposition.

With Lemma 3.20, we can, analogously to Alber et al. [8], partition the vertices in $G_1 \setminus V(\mathfrak{R})$ into three categories:

1. vertices in $\mathfrak{D}$,
2. degree-one vertices in $N^{G_1}(\mathfrak{D})$, and
3. vertices in $N_2^{G_1}(v)$ for some $v \in \mathfrak{D}$.

Clearly, the number of vertices in the first two categories is $O(\gamma(G))$. In the following, we show that the number of vertices of the third category that survive Phase 3 of the kernelization (recall the kernelization phases stated on page 86) is also in $O(\gamma(G))$. To do this, we need the following lemma.

Lemma 3.21. *Let $(\{v, w\}, U, \emptyset)$ be a raw diamond in G_1 such that $v \in \mathfrak{D}$ and $U \cap V(\mathfrak{R}) = \emptyset$. Then, $|U \cap V(G_2)| \in O(|U_B|)$.*

Proof. Lemma 3.14 is very useful to prove the claim. However, Lemma 3.14 requires that both v and w are in G_2. We consider the cases individually:

Case 1: $\{v, w\} \subseteq V(G_2)$. We first show that $U \cap \mathfrak{D} = \emptyset$. For the sake of contradiction, suppose that there is a vertex $u' \in U \cap \mathfrak{D}$. Then, by maximality of $\mathfrak{R}$, there is a region $\mathcal{R}(v, u') \in \mathfrak{R}$ (possibly only consisting of a single edge), contradicting $U \cap V(\mathfrak{R}) = \emptyset$. Hence, $V(\mathcal{R}(v, w)) \cap \mathfrak{D} \subseteq \{v, w\}$ and, thus, Lemma 3.14 implies that $|U \cap V(G_2)| \leq 5|U_B| + 3$.

Case 2: $\{v, w\} \not\subseteq V(G_2)$. Then, there is a vertex $x \in \{v, w\} \setminus V(G_2)$. First, consider the case in which x is deleted by Algorithm 3.2. However, since Algorithm 3.2 implements this rule and only deletes low-degree vertices, we can conclude that there are at most 78 vertices in $U \cap V(G_2)$.

Next, suppose that x is deleted by Reduction Rule 3.1. Then, there is an intermediate graph G' such that there is a vertex $v' \in V(G')$ with $x \in N_3^{G'}(v')$ and $U \cap V(G') \subseteq N^{G_1}[v']$. If $v' \notin \{v, w\}$, then by Lemma 3.8(1), it follows that $|U \cap V(G')| \leq 2$. Otherwise, $v' \in \{v, w\}$ and, obviously, $v' \neq x$. If $x = v$, then $v' = w$ and, hence, Reduction Rule 3.1 attaches a dummy vertex to w and by Observation 3.3, w is not an inner vertex of a region in $\mathfrak{R}$. However, since $U \cap V(\mathfrak{R}) = \emptyset$, we can add a region containing the raw-diamond $(\{v, w\}, U, \emptyset)$ to $\mathfrak{R}$, contradicting its maximality.

In the following, let $x = w$ and, thus, $v' = v$. Observe that $U \cap N_1^{G'}(v) = \emptyset$ because $w \in N_3^{G'}(v)$ and each vertex in U is adjacent to w. Furthermore, since $\{v, w\} \cap N_2^{G'}(v) = \emptyset$, it is clear that no vertex of $N_1^{G'}(v)$ is in $(\{v, w\}, U, \emptyset)$. Since each vertex in $N_2^{G'}(v)$ is adjacent to a vertex in $N_1^{G'}(v)$, it follows that $U \cap N_2^{G'}(v) \subseteq U_B$. Finally, observe that Reduction Rule 3.1 removes the set $U \cap N_3^{G'}(v)$ along with w. Hence, $|U \cap V(G_2)| \leq |U_B|$. □

In the following, we make use of the previous lemmas to prove Proposition 3.2.

Proposition 3.2. $|(V(G_1) \setminus V(\mathfrak{R})) \cap V(G_2)| \in O(\gamma(G))$.

Proof. This proof is very similar to the proof of Proposition 2 of Alber et al. [8]. First, let $Y := (V(G_1) \setminus V(\mathfrak{R})) \cap V(G_2)$ and recall the statement of Proposition 3.2: $|Y| \in O(\gamma(G))$. We will split Y into three parts and show that each of them contains at most $O(\gamma(G))$ vertices.
Since G_1 is reduced with respect to Reduction Rule 3.1, we can upper bound the size of $X := \bigcup_{v \in \mathfrak{D}} N_3^{G_1}(v)$ by $O(\gamma(G))$. Furthermore, recall that $|\mathfrak{D}| \leq 2\gamma(G)$.
It remains to upper bound the size of $Y \setminus (X \cup \mathfrak{D})$. Let $u \in V(G_1) \setminus (V(\mathfrak{R}) \cup X \cup \mathfrak{D})$. Since $u \notin X \cup \mathfrak{D}$, its degree in G_1 is at least two. Thus, by Lemma 3.20 there is a vertex $v \in \mathfrak{D}$ with $u \in N_2^{G_1}(v)$. Alber et al. [8, Proposition 2] showed that all vertices in $N_2^{G_1}(v) \setminus V(\mathfrak{R})$ can be partitioned into so-called *simple* regions in G_1 of the form $\mathcal{R}(v, y)$ such that $y \in N_1^{G_1}(v)$ and $\mathcal{R}(v, y) \subseteq \{v, y\} \cup (N_2^{G_1}(v) \setminus V(\mathfrak{R}))$. Furthermore, in total, there are at most $O(|\mathfrak{D}|)$ of these regions for all $v \in \mathfrak{D}$.[24]
By definition of simple regions [8, Definition 6], all vertices in $\mathcal{R}(v, y)$ are adjacent to v and y, it follows that with $U := \mathcal{R}(v, y) \setminus \{v, y\}$ the triple $(\{v, y\}, U, \emptyset)$ constitutes a raw diamond. Moreover, because each region has at most six boundary vertices, it follows that $|U_B| \leq 4$. By the argumentation of Alber et al. [8, Proposition 2], in order to show $|Y| \in O(|\mathfrak{D}|)$ it is sufficient to prove $|U \cap V(G_2)| \in O(1)$. This, however, follows from Lemma 3.21. □

Since we have shown in Section 3.4.2 that all regions of our $\mathfrak{D}$-region decomposition $\mathfrak{R}$ of G_1 have constant size in G_3 and, by choice of $\mathfrak{R}$, we know that $|\mathfrak{R}| \in O(\gamma(G))$, we have only $O(\gamma(G))$ vertices in regions. In this section, we showed that there are only $O(\gamma(G))$ vertices outsize of regions of $\mathfrak{R}$ in G_1 that are also in G_3. Since the number of additional (dummy) vertices of G_3 is in $O(\gamma(G))$, we conclude that G_3 contains only $O(\gamma(G))$ vertices, yielding Theorem 3.1 (see page 88).

Theorem 3.1. *On planar graphs, a linear-size problem kernel for* Dominating Set *is computable in linear time.*

3.5 Conclusion

Our work is meant to provide a first case study, using the well-known kernelization of Dominating Set on planar graphs, on how known kernelization algorithms can be tuned for better time performance by carefully analyzing the interaction and costs of the underlying data reduction rules. Clearly, on the practical side it is important to further improve on the upper bound for the kernel size to be achieved in linear time.
Informally speaking, the analysis of our problem kernel shows that "knowing when to stop" may be a source of performance gain. This is particularly true for

kernelization algorithms since faster algorithms with worse kernel bounds can be combined with slower algorithms yielding better kernel bounds in order to achieve an overall improvement. However, an empirical study in the spirit of algorithm engineering remains to be done.

As to future challenges, there are a lot of possibilities in revisiting known kernelization results and re-engineering them in terms of algorithmic efficiency beyond "polynomial-time" computability. A canonical example is Dominating Set on claw-free graphs, which was recently shown to have a $O(k^4)$-vertex kernel [118]. Furthermore, a recently shown linear-time kernel for Hitting Set with constant set size parameterized by the solution size k [18] motivates looking at similar problems that admit a polynomial-size kernel [177].

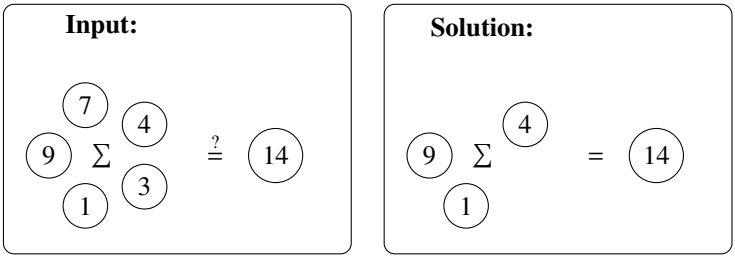
In this chapter, we developed a linear-time computable problem kernel of size $O(\gamma(G))$ for Dominating Set on planar graphs. For other problems, it might be helpful to alleviate the quest from $O(n)$-time to almost linear-time such as $p(k) + O(n)$ or $p(k) \cdot n$, where p is a polynomial solely depending on a parameter k. For instance, Chor et al. [49] presented such an "almost linear-time" kernelization for a variant of the Clique Covering problem, asking whether it is possible to cover all vertices of a graph with k cliques. It is a challenge for future research to investigate the complexity classes of fixed-parameter tractable problems that allow for kernels of *any size* computable in *(almost) linear running time*.

Chapter 4

Preprocessing Beyond Kernelization

In this chapter, we demonstrate how preprocessing can improve theoretical running time bounds of exact algorithms, even outside the scope of kernelization. A previous example of a preprocessing-based exact algorithm was sketched for the Subset Sum problem [173, 200].

Subset Sum
Input: A set S of n integers and an integer k.
Question: Is there a subset $I \subseteq S$ such that $\sum_{i \in I} i = k$?



Clearly, Subset Sum can be solved in $O(2^n \cdot n)$ time by checking the sum of all subsets of S. However, the following preprocessing-based algorithm runs in $O(2^{n/2} \cdot n)$ time.

Step 1: Let $\tilde{s}$ denote the median of S. Compute the set S_1 of all integers that can be obtained by summing up integers in S that are smaller than $\tilde{s}$ and let S_2 denote the set of all integers that can be obtained by summing up integers in S that are larger than $\tilde{s}$. Formally,

$$S_1 := \{\ell : \exists_{I \subseteq S} \sum_{i \in I} i = \ell \wedge \forall_{i \in I} i \leq \tilde{s}\}$$

$$S_2 := \{\ell : \exists_{I \subseteq S} \sum_{i \in I} i = \ell \wedge \forall_{i \in I} i > \tilde{s}\}$$

Step 2: Sort S_1 using merge sort.
Step 3: For each $\ell \in S_2$ determine whether $k - \ell \in S_1$ using binary search.

Partitioning the solution set I into $I_1 := \{i : i \in I \wedge i \leq \tilde{s}\}$ and $I_2 := I \setminus I_1$, it is clear that S_2 contains $\ell = \sum_{i \in I_2} i$ and S_1 contains $k - \ell = \sum_{i \in I_1} i$ and, therefore, the solution I is found by the algorithm. Since $|S_1|, |S_2| \leq 2^{n/2+1}$ and, thus, $\log |S_1| \leq n/2 + 1$, each step can be performed in $O(2^{n/2} \cdot n)$ time. This approach, however, has two downsides. First, it is inherent to number problems and therefore not applicable for problems on graphs. Second, the preprocessing routine itself takes exponential time, excluding it from combination with polynomial-time algorithms such as approximation or heuristics. An example of using preprocessing to obtain fixed-parameter tractability without supplying a kernel was recently given by Becker et al. [14], who combined reduction rules and branching to show fixed-parameter tractability for a generalization of Vertex Cover and Feedback Vertex Set called c-Pumpkin Hitting.
Here, we consider the Weighted Multigraph Eulerian Extension problem and demonstrate a polynomial-time preprocessing enabling us to prove fixed-parameter tractability. The problem asks for a minimum weight set of arcs to add to a given directed multigraph to make it Eulerian. We show that Weighted Multigraph Eulerian Extension is equivalent to the Rural Postman problem, which asks for a tour in a given directed graph visiting some specific arcs. Rural Postman is fundamental in arc routing with a long list of real-world applications [76, 67]. We develop a dynamic programming algorithm for the Weighted Multigraph Eulerian Extension problem (or, equivalently, the Rural Postman problem) running in $O(n^k 2^k \cdot n^4)$ time and show how adding an intuitive polynomial-time preprocessing improves the running time to $O(4^k \cdot n^3)$ time. Finally, incorporating observations granted by the preprocessing routine, we show a running time of $O(4^{b+2c} \cdot n^3)$ with b denoting the "imbalance" and c denoting the number of connected components of the input multigraph. The preprocessing does not constitute a problem kernel and we even demonstrate that Weighted Multigraph Eulerian Extension is unlikely to admit a polynomial-size problem kernel even with respect to the combined parameter $k + c + b$.

4.1 Introduction to Eulerian Extension

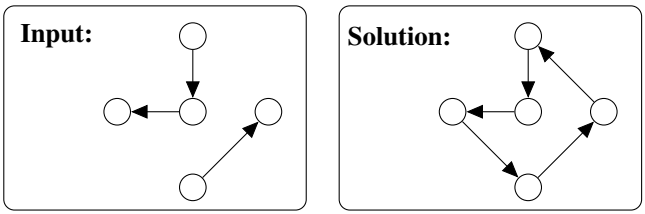
Edge modification problems in graphs have many applications and are well-studied in algorithmic graph theory [34, 158]. The corresponding minimization problems ask to modify as few (potential) edges as possible such that an input graph is transformed into a graph with a desired property. Most studies in this context relate to undirected graphs whereas we are aware of only few studies of

"arc modification" problems on directed graphs. Two examples are the works on the Transitivity Editing problem that asks whether a digraph can be made transitive[25] by a given amount of arc modifications [198] and the well-known Feedback Arc Set problem [42, 107, 16].

Here, we study the NP-hard problem of making a digraph Eulerian by arc additions. Following previous work [120], we call this "extension" problem. In the graph modification context, this is also known as "completion" or "addition" problem.

A digraph is called *Eulerian* if it contains an oriented cycle visiting every arc exactly once. An *Eulerian extension* is a set of arcs to add to a digraph so that it becomes Eulerian. The corresponding decision problem reads as follows.

Eulerian Extension
Input: A digraph G and an integer k.
Question: Is there an Eulerian extension $\mathcal{E}$ for G with $|\mathcal{E}| \leq k$?

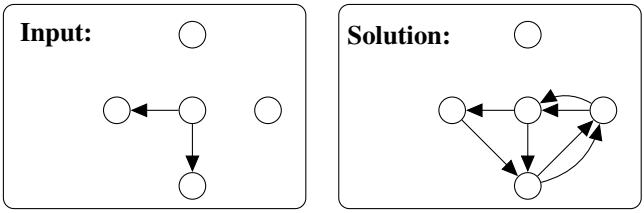


Variants of Eulerian Extension include Weighted Eulerian Extension, where an additional weight function $\omega : V \times V \to \mathbb{N}$ is given[26] that assigns weights to non-arcs and the solution is required to have total weight at most $\omega_{\max}$. There are also unweighted and weighted *multigraph* variants (where parallel arcs are allowed in the input and output) referred to as Multigraph Eulerian Extension and Weighted Multigraph Eulerian Extension, respectively. The main focus of this chapter lies with the latter of these problems:

[25] A digraph G is called *transitive* if the existence of the arcs (u, v) and (v, w) in G implies the existence of (u, w) in G.

[26] We assume that $0 \in \mathbb{N}$.

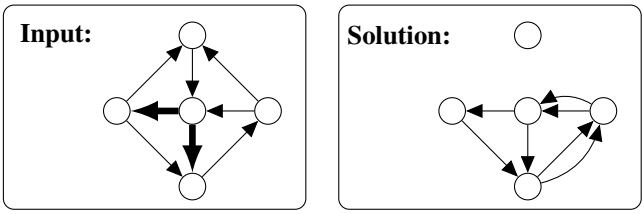
Weighted Multigraph Eulerian Extension
Input: A directed multigraph $M = (V, A)$, a weight function $\omega : V \times V \to \mathbb{N}$, and an integer $\omega_{\max}$.
Question: Is there an Eulerian extension $\mathcal{E}$ for M with $\sum_{a \in \mathcal{E}} \omega(a) \leq \omega_{\max}$?



The problem has applications in scheduling where it generalizes the three-machine flowshop problem with "no-wait" constraint that aims at minimizing the number of interruptions (denoted by F3|nwt|$\mathcal{G}$ in standard scheduling notation [102]). This flowshop problem rose from an application in steel manufacturing [120]. Here, a special case of Weighted Multigraph Eulerian Extension with restricted weight function is used.

Eulerian extension problems are closely related to the well-known Chinese Postman problem (for surveys, refer to Eiselt et al. [75, 76] and the book of Dror [67]), where the goal is to find a shortest walk that visits all arcs of a given directed graph, and the more general Rural Postman problem [76, 140] that asks for a shortest walk covering at least a given set of arcs. These given arcs are called "required arcs". Although Rural Postman can be defined for undirected graphs, we focus on its directed version. For Rural Postman, we allow requiring to visit some arcs multiple times, that is, we allow the required arcs to form a multiset.

Rural Postman
Input: A digraph $G = (V, A)$, a multiset R of "required" arcs of A, a weight function $\omega : A \to \mathbb{N}$, and an integer $\omega_{\max} \geq 0$.
Question: Is there a closed walk W in G such that W visits all arcs in R and the total weight of W is at most $\omega_{\max}$?



Note that, if $R = A$, then Rural Postman degenerates to the Chinese Postman problem [36, 72, 75, 153].
Rural Postman has numerous applications that range from vehicle routing problems like street sweeping, snow plowing, garbage collection, road sanding, or bus routing to parking meter collection and electrical meter reading [67, 76, 138, 140]. As we will see in Section 4.2, Rural Postman is equivalent to Weighted Multigraph Eulerian Extension, implying both problems are suitable to model these applications. Furthermore, this equivalence implies that the NP-hardness of Weighted Multigraph Eulerian Extension directly follows from the known NP-hardness result for Rural Postman [140] (in fact, Rural Postman is one of the prominent NP-complete problems featured by Garey and Johnson [100]). Moreover, the fact that Rural Postman is solvable in polynomial time if the required arcs form a weakly connected component [97] directly implies that Weighted Multigraph Eulerian Extension is solvable in polynomial time if the input is weakly connected.

Related Problems and Previous Work. The unweighted and undirected Eulerian extension problems for graphs and multigraphs were already discussed in 1977 by Boesch et al. [31], who developed a linear-time algorithm for the multigraph case and a matching-based algorithm for the graph case. Later, Lesniak and Oellermann [141] surveyed undirected Eulerian graphs, including characterizations of Eulerian (multi-)graphs and graphs that can be extended to Eulerian graphs. Recently, Höhn et al. [120] initiated a study of Eulerian extension problems applied to sequencing problems in scheduling. They showed that special cases of Weighted Multigraph Eulerian Extension with weight function $\omega : V \times V \to \{0, 1\}$ are NP-complete. In particular, they considered vertex sets $V \subseteq \mathbb{R}_0^+ \times \mathbb{R}_0^+$ and for each $u = (u_x, u_y)$ and $v = (v_x, v_y)$ they set

$$\omega((u, v)) := \begin{cases} 0, & \text{if } u_x \leq v_x \text{ and } u_y \leq v_y \\ 1, & \text{otherwise.} \end{cases}$$

Höhn et al. [120] call this problem Two-Dimensional Eulerian Extension.
To the best of our knowledge, Weighted Multigraph Eulerian Extension has not been considered directly in the literature so far. However, there is an arsenal of results for the (equivalent) Rural Postman problem, many of which are based on results for the Traveling Salesman problem (see Section 1.2.4, page 15). Rural Postman was shown to be NP-hard [140] by reducing the NP-hard Traveling Salesman problem with unit weights to it. If edge weights may be zero (or arbitrarily small), then it is even NP-hard to approximate Traveling Salesman to within any constant factor [100], and so is approximating Rural Postman. However,

certain restrictions of Rural Postman allow for polynomial-time constant-factor approximations. In particular, Frederickson [97] and Jansen [126] presented algorithms that compute postman tours that are at most 50% longer than optimal tours. Both algorithms are based on an $O(n^3)$-time factor-3/2 approximation algorithm of Christofides [50] for Traveling Salesman and, therefore, require that the input respects the triangle inequality.[27]

From a parameterized perspective, little is known about Rural Postman. An implicit result can be obtained by combining a transformation of Rural Postman to Traveling Salesman by Pearn et al. [166] with the previously mentioned reduction of Hamiltonian Cycle to Rural Postman. This yields a problem kernel containing $6|R|$ vertices for Rural Postman (recall that R denotes the required arcs in the input instance). Combining the transformation of Rural Postman to Traveling Salesman with the well-known dynamic programming algorithm by Held and Karp [115] gives an algorithm solving Rural Postman in $O(8^{|R|} \cdot |R|^2 + n^3)$ time. Pondering the NP-hardness proof [140] of Rural Postman, one quickly notices that it produces very specific instances. Notably, the number c of connected components in (V, R) is large. Indeed, Orloff [164] concluded that "the critical factor is the number of disconnected components." While Frederickson [96] gave an algorithm that runs in $O(n^{2c+1}/c!)$ time, the question whether Rural Postman is fixed-parameter tractable with respect to c was implied repeatedly [97, 164].

Regarding other modification operations, Cai and Yang [38] considered Eulerian subgraphs of undirected graphs. They presented a color-coding based randomized algorithm for computing a k-edge Eulerian subgraph of a given graph and stated that the derandomized algorithm runs in $2^{O(k)} \cdot nm \log n$ time. They also showed that the problem of finding k vertices to delete to make a given graph Eulerian is $W[1]$-hard with respect to k. In recent work, Fomin and Golovach [90] continued in this line of research, answering open questions about dual parameterizations for parameterized problems introduced by Cai and Yang [38]. Furthermore, Fomin and Golovach [90] proved that both, the problem of computing a k-edge Eulerian subgraph and the problem of computing a k-vertex Eulerian subgraph are unlikely to admit polynomial-size problem kernels with respect to k. Cygan et al. [55] showed that the problem of deleting k arcs (edges) in order to obtain an Eulerian graph is NP-hard and fixed-parameter tractable with respect to k and is unlikely to admit a problem kernel whose size is polynomial in k. Cechlárová et al. [41, 40] examined a similar graph modification problem. There, the task is to delete a minimum number of arcs of a given digraph such that each strongly connected component of the resulting digraph is Eulerian. Apart from NP-hardness, they

[27] The input is required to be a complete graph and for each three vertices u, v, and w, the inequality $\omega(\{u, v\}) + \omega(\{v, w\}) \geq \omega(\{u, w\})$ holds.

	weighted, weakly connected	unweighted
undir. graph	$O(n^3)$ (Theorem 4.1)	$O(\overline{m}\sqrt{n})$ (Theorem 4.2)
undir. multigraph	$O(n^3)$ (Corollary 4.1)	$O(n+m)$ (Prop. 4.5)
dir. graph	$O(\overline{m}^2 + n\overline{m}\log n)$ (Prop. 4.2)	$O(\overline{m}(n+\overline{m}))$ (Prop. 4.4)
dir. multigraph	$O(n^3 \log n)$ (Prop. 4.3)	$O(n+m)$ (Prop. 4.5)

Table 4.1: Polynomial-time solvable Eulerian extension problems. Here, n denotes the number of vertices in the input, m denotes the number of edges (arcs) in the input, and $\overline{m}$ denotes the number of edges (arcs) in the complement of the input (di)graph. The running times are under the assumption that the complement of the input is given in advance. If this is not the case, then running times involving $\overline{m}$ receive an additional summand $O(n^2)$. In general, weighted variants of Eulerian extension problems are NP-hard if the input (multi-)graph is not (weakly) connected [120, 140].

showed fixed-parameter tractability and intractability for some application-related parameters. Under the name Min-DESC, this problem was recently considered by Crowston et al. [54] who showed that, on tournaments, it is fixed-parameter tractable with respect to the number of arcs to delete.

Our Results. Our main achievement is to show that Weighted Multigraph Eulerian Extension is fixed-parameter tractable with respect to the parameter "minimum number k of extension arcs". In fact, given a multigraph M on n vertices and a weight function ω, our algorithm finds an Eulerian extension $\mathcal{E}$ with minimum total weight among all Eulerian extensions $\mathcal{E}'$ for M in time $O(4^k \cdot n^3)$. Using the above-mentioned equivalence, our algorithm implies fixed-parameter tractability for Rural Postman with respect to the parameter "number k of non-required arcs in the solution", which can be considered an "above guarantee" parameterization, since all solutions contain all required arcs of an instance. We briefly sketch a composition algorithm that implies that Weighted Multigraph Eulerian Extension is unlikely to admit a polynomial-size problem kernel with respect to the parameter k.

We present a number of polynomial-time solvable variants of Eulerian extension problems. In particular, we consider inputs that are connected or unweighted (see Table 4.1). This stands in contrast to Rural Postman, whose unweighted variant is NP-hard [140]. The results translate to Rural Postman in the following way. Unweighted inputs correspond to instances of Rural Postman in which all edges (arcs) are present. (Weakly) connected inputs correspond to instances of

Rural Postman in which the required edges (arcs) form a (weakly) connected component. Disallowing multisets as inputs corresponds to disallowing using an edge (arc) more than once in a solution for Rural Postman.
Our polynomial-time algorithms extend known results for special cases of the Weighted Multigraph Eulerian Extension problem where the weight function only assigns values zero and one [120] (for these variants, only NP-hardness was shown so far; we provide a first algorithmic result for this case) and Rural Postman, for which mainly approximation, heuristic, and some polynomial-time algorithms for special cases were known [76, 97].

Organization of the Chapter. After fixing some problem-specific notation and making basic observations, we examine polynomial-time algorithms for special cases of Eulerian extension problems in Section 4.3. We present our dynamic programming algorithm for Weighted Multigraph Eulerian Extension and analyze its running time in Section 4.4. Then, we add a polynomial-time preprocessing to the algorithm, leading to the main result of this chapter in Section 4.5. We complement this result in Section 4.6 by proving that Weighted Multigraph Eulerian Extension is unlikely to admit a polynomial-size problem kernel. Finally, Section 4.7 concludes with a review on open questions and future work.

4.2 Problem-Specific Notation, Basic Observations

Let $G = (V, A)$ be a directed graph or multigraph (that is, a graph with parallel arcs allowed)—we also use the letter M to refer to multigraphs. The set of connected components of G is denoted by $\mathcal{C}_G$. For an arc set $\mathcal{E}$ and some arc a, we abbreviate $\mathcal{E} \cup \{a\}$ to $\mathcal{E} + a$. Likewise, $G + \mathcal{E}$ and $G + a$ denote the results of adding the arc set $\mathcal{E}$ and the arc a, respectively, to G.
An *Eulerian cycle* in a directed (multi-)graph G is a directed cycle that visits all arcs of G exactly once (possibly visiting each vertex more than once). If such a cycle exists, then we call G *Eulerian*. We call a (multi-)set $\mathcal{E} \subseteq V \times V$ an *Eulerian extension* for G if $(V, A \cup \mathcal{E})$ is Eulerian. Furthermore, $\mathcal{E}$ is called *optimal* if there is no Eulerian extension of less total weight for G, where the total weight of $\mathcal{E}$ is $\sum_{a \in \mathcal{E}} \omega(a)$ for a weight function $\omega : V \times V \to \mathbb{N}$. Note that, in this context, the sum is over all elements of the multiset $\mathcal{E}$, including identical elements. Hence, if arc a occurs twice in $\mathcal{E}$, then the weight of a is counted twice in the sum. Likewise, the cardinality $|\mathcal{E}| := \sum_{a \in \mathcal{E}} 1$ counts identical elements individually. For sets or multisets A of arcs, we abbreviate $\omega(A) := \sum_{a \in A} \omega(a)$. A *walk* W in G is a sequence of arcs of A such that each arc starts in the end vertex of the previous arc. Since each

arc can occur multiple times in a walk, we also consider walks as multisets of arcs. If the sequence W starts in the same vertex as it ends, then we call W *closed*. The *imbalance* of a vertex v is

$$\text{imbal}(v) := \text{indeg}(v) - \text{outdeg}(v).$$

Specifically, let $\mathcal{I}_G^+$ ($\mathcal{I}_G^-$) denote the multiset of vertices v of G for which $\text{imbal}(v) > 0$ ($\text{imbal}(v) < 0$). Herein, each vertex v is contained $|\text{imbal}(v)|$ times in the multiset. In an undirected graph, we define the imbalance $\text{imbal}(v)$ of a vertex v to be one if the number of its neighbors is odd and zero otherwise. For both directed and undirected (multi-)graphs G, vertices v of G with $\text{imbal}(v) = 0$ are called *balanced*, while all other vertices of G are called *imbalanced*, with $\mathcal{I}_G := \mathcal{I}_G^+ \uplus \mathcal{I}_G^-$ denoting the multiset of imbalanced vertices of G. We call $|\mathcal{I}_G|/2 = |\mathcal{I}_G^+| = |\mathcal{I}_G^-|$ the *imbalance* of G. Connected components in C_G that do not contain imbalanced vertices are called *balanced*. We refer to the set of all balanced components of G by $C_G^{\text{bal}} \subseteq C_G$. With the concept of vertex balance, we can state a well-known characterization of Eulerian graphs and multigraphs that helps us prove a (multi-)graph to be Eulerian.

Lemma 4.1 (Folklore)**.** *A (directed) (multi-)graph is Eulerian if and only if all edges (arcs) are in the same connected component and all vertices are balanced.*

Eulerian Extension and Related Problems. In the most general problem that we study, we have weights and allow the input and output to be multigraphs. Since multigraphs allow the presence of parallel arcs, we may also add arcs that are already present in the input. If we restrict the problem to digraphs, that is, we prohibit parallel arcs in both the input and the resulting digraph, then we arrive at the Weighted Eulerian Extension problem. Both Weighted Multigraph Eulerian Extension and Weighted Eulerian Extension are also considered in their unweighted versions, where all arcs have weight one.
In this chapter, all variants of Eulerian Extension are parameterized by the minimum cardinality k of all Eulerian extensions with weight at most $\omega_{\max}$. More formally,

$$k := \min\{|\mathcal{E}| : \mathcal{E} \text{ is an Eulerian extension for } G \text{ and } \omega(\mathcal{E}) \leq \omega_{\max}\}. \tag{4.1}$$

Rural Postman is parameterized by q, the minimum number of "additional" arcs over all solutions. More formally,

$$q := \min\{|W| - |R| : W \text{ is a walk in } G \text{ visiting all arcs in } R\}. \tag{4.2}$$

Note that, for Rural Postman, q is a "stronger" parameter than the number of arcs in W, because $q \leq |W|$. This implies that all positive (algorithmic) results for

Rural Postman parameterized by q also hold for Rural Postman parameterized by $|W|$. Since all solutions guarantee to contain R, choosing q can be considered an above-guarantee parametrization [146, 147, 157] of Rural Postman.

Helpful Observations. We present observations that help us prove our results and give insights into the structure of the considered problems. First, observe that, over all vertices of a graph, the imbalance always adds up to zero, that is, for each "missing" incoming arc, there is also a "missing" outgoing arc.

Observation 4.1. *Let G be a directed (multi-)graph. Then $\sum_{v \in V(G)} \text{imbal}(v) = 0$ and, equivalently, $|\mathcal{I}_G^+| = |\mathcal{I}_G^-|$.*

In undirected graphs and multigraphs, the sum over all imbalances is even. Observation 4.1 also be applies to connected components. Next, observe that an undirected Eulerian graph cannot have bridges since a walk visiting all edges would have to cross the bridge, effectively cutting off its way back to the starting point.

Observation 4.2. *Let G be a directed or undirected Eulerian (multi-)graph. Then there is no bridge in G.*

Next, consider an Eulerian extension $\mathcal{E}$ of some directed (multi-)graph G. Clearly, for each occurrence of a vertex v in $\mathcal{I}_G^+$, there is an arc outgoing of v in $\mathcal{E}$. Moreover, for each balanced connected component of G, there must be at least one arc in $\mathcal{E}$ that leaves this component. Considering that $|\mathcal{I}_G| = 2|\mathcal{I}_G^+|$, we can make the following observation.

Observation 4.3. *Let G be a directed or undirected (multi-)graph that is not Eulerian and let $\mathcal{E}$ be an Eulerian extension of G. Then $|\mathcal{I}_G|/2 + |\mathcal{C}_G^{bal}| \leq |\mathcal{E}|$.*

In the remainder of this section, we show an important relation between Rural Postman and Weighted Multigraph Eulerian Extension. An example illustrating this relation is shown in Figure 4.1.

Proposition 4.1. Rural Postman *and* Weighted Multigraph Eulerian Extension *are parameterized equivalent.*

Proof. The idea to show the claim is to identify the required arcs of the Rural Postman-instance with the arcs of the input multigraph for Weighted Multigraph Eulerian Extension. Furthermore, identify non-arcs in the Rural Postman-instance with non-arcs of weight ∞ in the Weighted Multigraph Eulerian Extension-instance. Note that parallel arcs carried into the Rural Postman instance

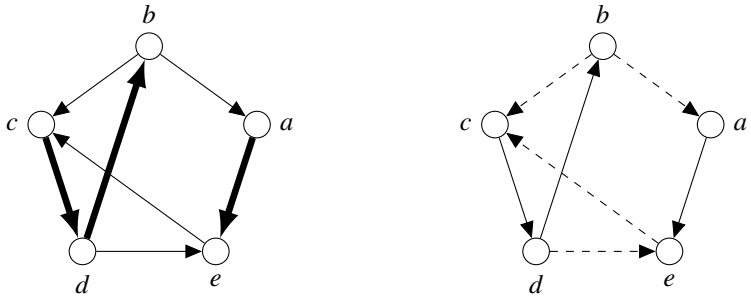


Figure 4.1: An illustration of equivalent instances of Rural Postman and Weighted Multigraph Eulerian Extension, respectively. Left: Instance of Rural Postman with required arcs drawn in bold. Right, a corresponding instance of Weighted Multigraph Eulerian Extension, where the dashed arcs are not present in the graph but have finite weight. Arcs that are not drawn have infinite weight. An exemplary postman tour visiting all required arcs would be (c, d, b, a, e, c). The corresponding Eulerian extension is $\{(b, a), (e, c)\}$. Note that the weight of the Eulerian extension equals the weight of the postman tour minus the weight of the required arcs.

from the Weighted Multigraph Eulerian Extension-instance can be handled by subdividing them.

First, to establish that Rural Postman is parameterized reducible to Weighted Multigraph Eulerian Extension, we construct an instance $(M, \omega', \omega'_{\max})$ with parameter k from an instance $(G = (V, A), R, \omega, \omega_{\max})$ of Rural Postman with parameter q by setting $M := (V, R)$, $\omega'_{\max} = \omega_{\max} - \omega(R)$, $k := q$, and

$$\omega'(a) := \begin{cases} \omega(a) & \text{if } a \in A, \\ \infty & \text{otherwise.} \end{cases}$$

Since the weight functions are basically equal, "the weight" of some arc set may refer to either ω or ω'. Let $\mathcal{E}$ denote a solution for $(M, \omega', \omega'_{\max})$. Then, by definition of Eulerian graphs, $W := \mathcal{E} \cup R$ is a walk of weight at most $\omega'_{\max} + \omega'(R)$ that visits all arcs in R. Furthermore, if we have a solution W for $(G, R, \omega, \omega_{\max})$, then $\mathcal{E} := W \setminus R$ is an Eulerian extension of weight at most $\omega_{\max} - \omega(R) = \omega'_{\max}$ for M with $|\mathcal{E}| \leq q = k$.

Second, we show that Weighted Multigraph Eulerian Extension is parameterized reducible to Rural Postman. Given an instance $(M = (V, A'), \omega', \omega'_{\max})$ of Weighted Multigraph Eulerian Extension with parameter k, we construct an

instance $(G, R, \omega, \omega_{\max})$ of Rural Postman with parameter q by setting $G := (V, V \times V)$, $R := A'$, $\omega := \omega'$, $\omega_{\max} := \omega'_{\max} + \omega'(R)$, and $q := k$. Note that parallel arcs in M can be handled by subdivision, that is, if the arc (u, v) appears twice in A' then we introduce a new vertex w into M and replace one of the arcs (u, v) with the two arcs (u, w) and (w, v) with $\omega((u, w)) = 0$ and $w((w, v)) = w((u, v))$.
Analogously to the first part of the proof, a solution $\mathcal{E}$ for $(M, \omega', \omega'_{\max})$ implies a solution $W = \mathcal{E} \cup R$ for $(G, R, \omega, \omega_{\max})$ and vice versa. □

4.3 Polynomial-Time Solvable Cases

As discussed in the introduction of this thesis, familiarity with polynomial-time special cases often helps construct parameterized algorithms with data reduction rules like Reduction Rule 1.3. Therefore, we briefly examine polynomial-time solvable cases of Eulerian Extension, generalizing previously known cases and giving exact running times for special cases that were just stated to be "polynomial-time solvable". More precisely, we present results for all polynomially solvable combinations of the properties "connected" and "weighted". First, we consider weighted variants of Eulerian extension problems if the input (multi-)graph is connected (Section 4.3.1). Then, we consider the unweighted variant and allow disconnected (multi-)graphs (Section 4.3.2).
All running times are given as functions in the number n of vertices in the input, the number m of edges (arcs) in the input, and the number $\overline{m}$ of edges (arcs) in the complement of the input graph. To allow focus on the algorithmic results, we assume that the complement of the input is given in advance. If this is not the case, then running times involving $\overline{m}$ receive an additional summand $O(n^2)$. We refer to Table 4.1 for an overview of the results of this section.

4.3.1 Algorithms for Connected Weighted Variants

Keeping in mind that the disconnected versions of Weighted Eulerian Extension and Weighted Multigraph Eulerian Extension are NP-hard [120, 140], we provide polynomial-time algorithms for both problems in case of connected inputs. When considering directed inputs, "connected" always means "weakly connected".
The first result states that Weighted Eulerian Extension on connected, undirected graphs can be solved in polynomial time. Based on the notion of "T-joins", this can be derived from a result by Edmonds and Johnson [73]. For a graph G and a vertex-set T, a *T-join* is a set J of edges of G such that each vertex v of G

is incident to an odd number of edges in J if and only if $v \in T$. Edmonds and Johnson [73] showed a running time of $O(n^3)$ for solving the following problem.

MINIMUM-WEIGHT T-JOIN
Input: An undirected graph $G = (V, E)$, a set $T \subseteq V$, a weight function $\omega : E \to \mathbb{N}$, and an integer k.
Question: Is there a T-join of weight at most k for G?

Letting T be the set of imbalanced vertices in a connected undirected input graph G, solving MINIMUM-WEIGHT T-JOIN corresponds to solving WEIGHTED EULERIAN EXTENSION.

Theorem 4.1 ([73, 135])**.** WEIGHTED EULERIAN EXTENSION *on connected undirected graphs can be solved in* $O(n^3)$ *time.*

In order to apply the result of Edmonds and Johnson [73] to multigraphs, we show that there is a minimum-weight Eulerian extension in a connected undirected multigraph that does not use any edge more than once.

Lemma 4.2. *Let $M = (V, E)$ be a connected undirected multigraph, let $\omega : \binom{V}{2} \to \mathbb{N}$ be an edge-weight function and let $\mathcal{I}_M$ denote the imbalanced vertices of M. Let G be a clique on the vertex set V. Then, a minimum-weight $\mathcal{I}_M$-join in G is also a minimum-weight Eulerian extension $\mathcal{E}$ for M.*

Proof. Clearly, each $\mathcal{I}_M$-join in G is an Eulerian extension for M. However, to show that there is no Eulerian extension of less weight, we prove that, for each minimum-weight Eulerian extension that uses some edge twice, we can find a minimum-weight Eulerian extension that uses less edges twice. By iterating this argument, we can find a minimum-weight Eulerian extension that does not use any edge more than once and, hence, this Eulerian extension also constitutes a minimum-weight $\mathcal{I}_M$-join in G. Note that, since M is connected, $\mathcal{E}$ does not contain cycles.
In the following, assume that $\mathcal{E}$ contains a path between vertices u and v and a path between vertices x and y and the two paths share an edge $\{a, b\}$. Now, consider the following paths in G. The first path p_1 consists of the part $(u, \ldots, a)$ of the path between u and v and the reverse of the part $(x, \ldots, a)$ of the path between x and y. The second path p_2 consists of the reverse of the part $(b, \ldots, v)$ of the path between u and v and the part $(b, \ldots, y)$ of the path between x and y.
Obviously, the union of the sets of edges of p_1 and p_2 is a proper subset of the union of the sets of edges of the paths between u and v and x and y, respectively. Hence, replacing the paths in $\mathcal{E}$ accordingly does not increase the weight of $\mathcal{E}$ and decreases the number of edges used more than once. □

With Lemma 4.2, we can now state a version of Theorem 4.1 for connected undirected multigraphs.

Corollary 4.1. Weighted Multigraph Eulerian Extension *on connected undirected multigraphs can be solved in* $O(n^3)$ *time.*

It remains to show polynomial-time solvability for directed graphs and multigraphs. First, we present an algorithm for digraphs, which is then modified to work for directed multigraphs. The algorithm is based on computing network flows or matchings (refer to Ahuja et al. [5] for flow notations).

Proposition 4.2. Weighted Eulerian Extension *on connected digraphs can be solved in* $O(\overline{m}^2 + n\overline{m} \log n)$ *time.*

Proof. Consider an instance $(G, \omega, k, \omega_{\max})$ of Weighted Eulerian Extension, where G is a connected digraph, and a function imbal : $V(G) \to \mathbb{Z}$ measuring the imbalance of each vertex (see Section 4.2). Consider the flow network $\overline{G}$ (the complement graph of G) with supply determined by the function imbal (negative supply indicates demand), arc capacity one for each arc, and arc-costs determined by ω. Let f be an integral flow of value $1/2 \cdot \sum_{v \in V} |\text{imbal}(v)|$ in this network. Then, the set of arcs carrying f corresponds to an Eulerian extension for G and, thus, the minimum cost of such a flow is also the minimum cost of an Eulerian extension for G. Such a flow can be computed in $O(\overline{m}^2 + n\overline{m} \log n)$ time.[28] □

Next, we allow parallel arcs in the input and modify the algorithm described in the proof of Proposition 4.2. For a directed multigraph M let G_M be the complete digraph (containing all possible arcs) on the vertex set of M. Analogously to the proof of Proposition 4.2, we employ a min-cost flow algorithm on G_M with arc capacities ∞ and weights according to ω and supply determined by the function imbal. The uncapacitated version of the min-cost flow algorithm (running in $O(n^3 \log n)$ time [5]) can be used in this case.

Proposition 4.3. Weighted Multigraph Eulerian Extension *on connected directed multigraphs can be solved in* $O(n^3 \log n)$ *time.*

4.3.2 Algorithms for General Unweighted Variants

In this section, we provide polynomial-time algorithms for Eulerian Extension on various input (multi-)graphs. First, we state a previously known result for Eulerian Extension and show how algorithms of Section 4.3.1 help to solve the problem on connected digraphs. We then present an algorithm for disconnected

[28] See Exercise 10.17 of Ahuja et al. [5], a solution to which can be found on the web [6].

digraphs and an algorithm solving Eulerian Extension on directed and undirected multigraphs.
Eulerian extensions were already considered by Boesch et al. [31], who showed that, on undirected graphs, Eulerian Extension can be solved in polynomial time, even for disconnected graphs.

Theorem 4.2 ([31, 141])**.** Eulerian Extension *on undirected graphs can be solved in* $O(\overline{m}\sqrt{n})$ *time.*

Next, we extend this result to digraphs. Since Eulerian Extension is a special case of Weighted Eulerian Extension, we can solve Eulerian Extension for connected digraphs using the algorithm from the proof of Proposition 4.2 with a unit-weight version of the min-cost flow algorithm running in $O(\overline{m}(n+\overline{m}))$ time.[29]

Corollary 4.2. Eulerian Extension *on connected directed graphs can be solved in* $O(\overline{m}(n+\overline{m}))$ *time.*

This algorithm cannot handle multiple components. In the following, we present a more general algorithm that also allows us to solve the problem on disconnected digraphs (at the cost of increased running time). The algorithm starts with a min-cost max-flow in the complement digraph and locally modifies the arcs that carry flow in order to connect all components of the input digraph. For ease of presentation we split the following lemma concerning digraphs of diameter two[30] from the main correctness proof.

Lemma 4.3. *Let f be a min-cost max-flow in a unit-weight unit-capacity digraph G. Let F denote the arcs of G that carry flow and let $G - F$ have diameter two. Then there are no three consecutive arcs in F.*

Proof. Assume that F contains three consecutive arcs (u,v), (v,w), and (w,x). Since $G - F$ has diameter two, there is a vertex v' in $G - F$ such that (u,v') and (v',x) are in $G - F$. Hence, replacing (u,v), (v,w), and (w,x) with (u,v') and (v',x) in F yields a max-flow of less cost, contradicting our choice of F. □

The following algorithm receives a digraph $G = (V,A)$ and a vertex imbalance function $\text{imbal} : V \to \mathbb{Z}$ and computes a minimum-cost Eulerian extension for G, if it exists. The algorithm comprises three steps that modify the input digraph G. In Step i, we denote the current extension set by F_i.

[29] Combine the solution [6] for Exercise 10.17 of Ahuja et al. [5] with breadth-first search as shortest path algorithm.

[30] In a digraph, the *distance* of u to v is the number of arcs in a shortest path from u to v if such a path exists, and ∞ otherwise. The *diameter* of a digraph G is the maximum distance between any pair of vertices in G.

Step 1: Compute a min-cost max-flow f of value $|\mathcal{I}_G|/2$ (recall that $\mathcal{I}_G$ denotes the multiset of imbalanced vertices of G) in the complement graph $\overline{G}$ with supply determined by the function imbal (negative supply indicates demand), arc capacities one and arc weights one for each arc. If there is no such flow, then there is no Eulerian extension for G. Let F_1 denote the set of arcs that carry flow. Note that $G + F_1$ does not contain imbalanced vertices.

Step 2: Let F_2 denote the current extension set after Step 1. In this step, we modify F_2 such that all vertices that are imbalanced in G are in the same connected component of $G + F_2$. If this is already the case, then continue with Step 3. Otherwise, find two different connected components C_1 and C_2 containing vertices that are imbalanced in G. By Observation 4.2, there are arcs $(u, v) \in (C_1 \times C_1) \cap F_2$ and $(x, y) \in (C_2 \times C_2) \cap F_2$ whose removal does not disconnect C_1 or C_2. Replace (u, v) and (x, y) with (u, y) and (x, v) in F_2 and repeat Step 2. If $G+F_2$ is connected, then return F_2 as an optimal Eulerian extension for G.

Step 3: Let F_3 denote the current extension set after Step 1. Since $G + F_2$ is disconnected, $\overline{G + F_2} = \overline{G} - F_2$ has diameter two and, by Lemma 4.3, F_3 does not contain paths of length three. Let F_3^1 and F_3^2 denote the sets of paths of arcs in F_3 with length one and two, respectively. Join connected components of $G+F_2$ by first "rerouting" paths in F_3^2 (which can be done by replacing their middle vertex with a vertex in another connected component) if this operation decreases the number of connected components. When this is no longer possible, "split open" an arbitrary path in $G + F_3$ so that it additionally contains a vertex in another connected component.

Proposition 4.4. Eulerian Extension *on disconnected digraphs can be solved in* $O(\overline{m}(n + \overline{m}))$ *time.*

Proof. First, we show that the presented algorithm produces an optimal Eulerian extension for G. Let $\mathcal{E}$ denote the set of arcs that are returned by the algorithm above. After Step 1, $G + F_1$ does not contain imbalanced vertices and for each arc (u, v) removed from F_1 in later steps, a path from u to v is added. Hence, $G + \mathcal{E}$ does not contain imbalanced vertices. Since the algorithm only returns connected digraphs, $G + \mathcal{E}$ is also connected and, thus, Eulerian.
If the digraph is connected before splitting paths in Step 3, then $|\mathcal{E}| = |F_2|$. Note that no smaller Eulerian extension exists since this would imply a lower cost flow in $\overline{G}$, contradicting the fact that the cost of the flow corresponding to F_1 is minimum. Otherwise, for each path p of F_3^2, there is a different connected component C of the input digraph G such that p contains a vertex of C. Hence, at most $|\mathcal{C}_G^{\text{bal}}| - |F_3^2|/2$ arcs are added in Step 3, implying $|\mathcal{E}| = |F| + (|\mathcal{C}_G^{\text{bal}}| - |F_3^2|/2) =$

$|F_3^1| + |F_3^2|/2 + |C_G^{\text{bal}}|$. By definition of F_3^1 and F_3^2, $2|F_3^1| + |F_3^2| = |\mathcal{I}_G|$. Hence, $|\mathcal{E}| = |\mathcal{I}_G|/2 + |C_G^{\text{bal}}|$ and, by Observation 4.3, there is no smaller Eulerian extension than $\mathcal{E}$.

It is not hard to see that computing the min-cost max-flow in in Step 1 dominates the overall running time. This flow can be computed in $O(\overline{m}(n + \overline{m}))$ time [5]. □

Proposition 4.4 stands in contrast with Rural Postman being NP-hard for unweighted digraphs [140], which seems to be due to the fact that the input for Rural Postman may prohibit arcs by excluding them from the input digraph. It turns out that, if no arc is forbidden, that is, the input digraph is complete, then Rural Postman is solvable in polynomial time. More precisely, we can solve Multigraph Eulerian Extension for directed multigraphs M by the following straightforward greedy strategy much like the algorithm known for undirected multigraphs [31].

Step 1: For each connected component C of M, arbitrarily select a pair of vertices $(x_C, y_C) \in \mathcal{I}_C^+ \times \mathcal{I}_C^-$ if C is imbalanced, and $(x_C, y_C) \in \{(v, v) : v \in V(C)\}$ if C is balanced.

Step 2: Choose an arbitrary order $(C_1, C_2, \ldots, C_{|C_M|})$ of the connected components and add the arcs $(y_{C_1}, x_{C_2}), (y_{C_2}, x_{C_3}), \ldots, (y_{C_{|C_M|}}, x_{C_1})$, thus connecting all connected components in a circular manner.

Step 3: Greedily insert arcs between the remaining imbalanced vertices until all vertices are balanced.

Proposition 4.5. Multigraph Eulerian Extension *on directed and undirected multigraphs can be solved in* $O(n + m)$ *time.*

Proof. Boesch et al. [31] showed the claim for undirected multigraphs. We extend this to directed multigraphs. If the input multigraph M is connected, then we can skip Step 1 and Step 2. Clearly, after Step 2, the multigraph is connected, and after Step 3 all vertices are balanced. Hence, by Lemma 4.1, the set $\mathcal{E}$ of all inserted arcs is an Eulerian extension for M.

Next, we show that $\mathcal{E}$ is also optimal. Let $\mathcal{E}_2$ and $\mathcal{E}_3$ denote the arcs that are added in Step 2 and Step 3, respectively, and define X_2 and X_3 as the multisets that contain a vertex v as many times as there are arcs in $\mathcal{E}_2$ and $\mathcal{E}_3$, respectively, that are incident to v. Clearly, $|X_2| = 2|\mathcal{E}_2| = 2|C_M|$ and there are $2|C_M| - 2|C_M^{\text{bal}}|$ imbalanced vertices in X_2. Since the result of Step 3 is balanced, $|X_3| = |\mathcal{I}_M| - (2|C_M| - 2|C_M^{\text{bal}}|)$. Hence,

$$2|\mathcal{E}| = |X_2| + |X_3| = 2|C_M| + |\mathcal{I}_M| - (2|C_M| - 2|C_M^{\text{bal}}|) = |\mathcal{I}_M| + 2|C_M^{\text{bal}}|$$

and, by Observation 4.3, there is no smaller Eulerian extension than $\mathcal{E}$.

For the running time note that, in Step 1, we just have to scan each connected component, and in Step 2 and Step 3, each arc addition can be done in constant time. □

4.4 Tractability on Directed Multigraphs

In this section, we describe a dynamic programming algorithm solving Weighted Multigraph Eulerian Extension in $O(n^k 2^k \cdot n^4)$ time, where k denotes the cardinality of the sought Eulerian extension (more thoroughly defined in Section 4.2 on page 115). More precisely, given a multigraph M, a weight function ω, and some integer k, our algorithm finds an Eulerian extension $\mathcal{E}$ with minimum total weight over all Eulerian extensions $\mathcal{E}'$ for M with $|\mathcal{E}'| = k$. If k is not known in advance, then we employ a simple self-reduction that, given an instance $(M, \omega, \omega_{\max})$ of Weighted Multigraph Eulerian Extension, finds an Eulerian extension $\mathcal{E}$ for M with the following properties:

(i) the total weight of $\mathcal{E}$ is at most $\omega_{\max}$,
(ii) of all Eulerian extensions satisfying (i), $\mathcal{E}$ has minimum cardinality, and
(iii) of all Eulerian extensions satisfying (i) and (ii), $\mathcal{E}$ has minimum total weight.

The algorithm is developed in three steps. First, in Section 4.4.1, we transform the input into an instance of a modified problem called Black/Gray Weighted Multigraph Eulerian Extension. This problem has the advantage that a corresponding Eulerian extension has a particularly simple structure to be exploited by a dynamic programming algorithm. Then, in Section 4.4.2, we present such an algorithm for Black/Gray Weighted Multigraph Eulerian Extension. Finally, in Section 4.4.3, we put everything together to obtain an algorithm for Weighted Multigraph Eulerian Extension.

4.4.1 Transformation to a Weaker Variant

We introduce the Black/Gray Weighted Multigraph Eulerian Extension problem and show how it helps us solve Weighted Multigraph Eulerian Extension. For this, we picture Eulerian extensions as collections of paths between imbalanced vertices as stated in the forthcoming observation. This notion is fundamental for the algorithm for Black/Gray Weighted Multigraph Eulerian Extension that we present in this section. It is based on the fact that for each balanced vertex u, each Eulerian extension contains as many arcs outgoing of u as incoming to u.

Observation 4.4. *Let M be a directed multigraph and let $\mathcal{E}$ be an Eulerian extension for M. Then $\mathcal{E}$ can be decomposed into paths that start at a vertex in $\mathcal{I}_M^+$ and end at a vertex in $\mathcal{I}_M^-$ or start and end at the same vertex (cycles).*

Our idea to attack Weighted Multigraph Eulerian Extension is to use dynamic programming to construct the paths of an optimal Eulerian extension arc by arc. There are, however, a few obstacles to this approach. Assuming that no path contains two vertices of the same component proved helpful in overcoming these obstacles. Since this is not always the case, we modify the input multigraph in order to use it in the following, somewhat more technical problem, for which this assumption is valid.

Black/Gray Weighted Multigraph Eulerian Extension
Input: Two directed multigraphs $M_{\text{black}} = (V, A_{\text{black}})$ and $M_{\text{gray}} = (V, A_{\text{gray}})$ such that each connected component of M_{black} is either balanced or contains exactly two imbalanced vertices (one with imbalance 1, one with imbalance -1), a weight function $\omega : V \times V \to \mathbb{N}$, and an integer $\omega_{\max}$.
Question: Is there an Eulerian extension $\mathcal{E}'$ of weight at most $\omega_{\max}$ for $M := (V, A_{\text{black}} \cup A_{\text{gray}})$ such that in each component C_{black} of M_{black} there is exactly one tail of an arc in $\mathcal{E}'$ and exactly one head of an arc in $\mathcal{E}'$ (that is, $|(V(C_{\text{black}}) \times V) \cap \mathcal{E}'| = |(V \times V(C_{\text{black}})) \cap \mathcal{E}'| = 1$)?

Again, we can decompose a black/gray Eulerian extension $\mathcal{E}'$ into paths analogously to Observation 4.4. The advantage of Black/Gray Weighted Multigraph Eulerian Extension is that, in total, these paths visit each black component exactly once. The gray arcs (arcs in A_{gray}) are used to model the connectivity constraints given by the original Weighted Multigraph Eulerian Extension instance. We first describe how Weighted Multigraph Eulerian Extension can be solved using an algorithm for Black/Gray Weighted Multigraph Eulerian Extension and then present such an algorithm for Black/Gray Weighted Multigraph Eulerian Extension in Section 4.4.2.

To solve Weighted Multigraph Eulerian Extension using Black/Gray Weighted Multigraph Eulerian Extension we employ a transformation that, given an instance $(M, \omega, \omega_{\max})$ of Weighted Multigraph Eulerian Extension, computes and an instance $(M', \omega', \omega'_{\max})$ of Black/Gray Weighted Multigraph Eulerian Extension. In this transformation algorithm, we compute a bijection[31] $\mu \subseteq \mathcal{I}^-_M \times \mathcal{I}^+_M$ such that each two vertices related by μ are in the same connected component of M. We call such a bijection *component-respecting*. Since Observation 4.1 holds for each connected component, there is a component-respecting bijection μ for each instance $(M, \omega, \omega_{\max})$ of Weighted Multigraph Eulerian

[31] We say that μ is bijective if for all submultisets $X \subseteq \mathcal{I}^-_M$, it holds that $|\mu \cap (X \times \mathcal{I}^+_M)| = |X|$ and for all submultisets $X \subseteq \mathcal{I}^+_M$ it holds that $|\mu \cap (\mathcal{I}^-_M \times X)| = |X|$.

Extension. The transformation receives, additionally to the instance $(M, \omega, \omega_{\max})$, a function $\# : C_M \to \mathbb{N}$ indicating for each connected component C of M, the number of times it is visited by the sought solution. We denote the transformed instance by $\mathrm{tr}_{\#}(M, \omega, \omega_{\max}) := (M', \omega', \omega_{\max})$. See Figure 4.2for an example. The following describes the transformation algorithm.

Step 1: Compute a component-respecting bijection μ.

Step 2: Compute a new multigraph M' by creating $\#(C)$ copies of each connected component C of M. Construct ω' such that $\omega'(x', y') = \omega(x, y)$ for all vertices x, y of M and their copies x', y' in M'.

Step 3: For each component C of M, assign a copy C' of C to each pair $(v, w) \in \mu$ of imbalanced vertices of C and *isolate* (v, w) in C', that is, add all arcs in $(\mathcal{I}_C^- \times \mathcal{I}_C^+) \setminus \{(v, w)\}$ to C'. All copies that have not been assigned to an imbalanced pair are balanced completely in the above mentioned way. This assures that each copy of C contains at most one pair (v', w') of vertices and their imbalance is 1 and -1, respectively. Furthermore, each pair of imbalanced vertices in μ is represented in exactly one copy.

Step 4: For each component C of M, its copies are pairwisely connected by adding gray arcs. To this end, select any vertex v of C and add all possible arcs between all copies of v. Note that only copies of the same component of M are connected by gray arcs.

In the following, we show that the above transformation is correct. More specifically, it turns out that one can obtain a solution for an instance of Weighted Multigraph Eulerian Extension by transforming the input for all feasible $\# : C_M \to \mathbb{N}$ and solving these transformed instances of Black/Gray Weighted Multigraph Eulerian Extension. Indeed, we can reduce the number of functions # for which $(M, \omega, \omega_{\max})$ is to be transformed by imposing the following restrictions on #. Since a solution $\mathcal{E}$ for Black/Gray Weighted Multigraph Eulerian Extension visits each copy exactly once (each copy forms a black component), summing up $\#(C)$ over all connected components C of G must not exceed $|\mathcal{E}|$ $(= k)$ and since each copy is assigned to at most one pair of imbalanced vertices, each connected component C must have at least $|\mathcal{I}_C|/2$ copies. Thus, we need only consider functions of the form

$$\# : C_M \to \mathbb{N}, \text{ with } \sum_{C \in C_M} \#(C) \leq k \text{ and } \forall C \in C_M : \#(C) \geq |\mathcal{I}_C|/2 \,. \tag{4.3}$$

To prove the correctness of the described transformation, we show that finding a solution for an instance $(M, \omega, \omega_{\max})$ of Weighted Multigraph Eulerian Extension is equivalent to finding a solution for an instance $\mathrm{tr}_{\#}(M, \omega, \omega_{\max})$ of Black/Gray Weighted Multigraph Eulerian Extension for any fixed μ.

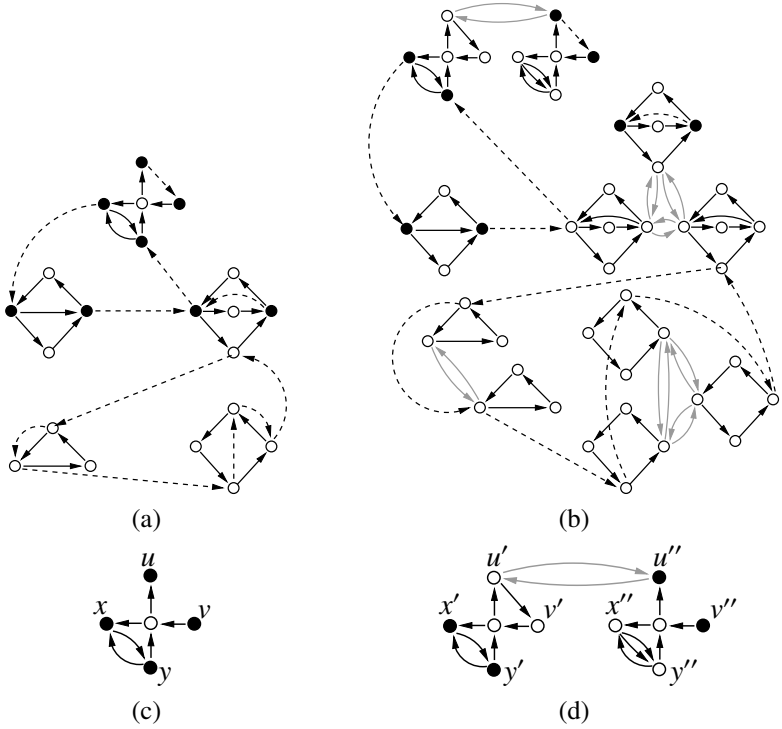


Figure 4.2: The picture shows a directed multigraph with an Eulerian extension containing eleven (dashed) arcs (a) being transformed to (b) by the transformation $\mathrm{tr}_{\#}$ described in Section 4.4.1. White vertices are balanced, black vertices are imbalanced. In (c) and (d), we take a closer look at the top connected component C of the graph depicted in (a) and its transformation. Here, $\#(C) = 2$, $\mu(v) = u$ and $\mu(y) = x$. Note that the pair (y'', x'') is isolated in the first copy and the pair (v', u') is isolated in the second copy. Finally, as described in the last step of the transformation, u' and u'' are connected by gray arcs.

Lemma 4.4. *The instance* $(M, \omega, \omega_{max})$ *is a yes-instance of* Weighted Multigraph Eulerian Extension *if and only if there is a function* $\# : C_M \to \mathbb{N}$ *complying with* (4.3) *such that* $\mathrm{tr}_{\#}(M, \omega, \omega_{max})$ *is a yes-instance of* Black/Gray Weighted Multigraph Eulerian Extension.

Proof. Throughout this proof, let $(M', \omega', \omega'_{\max}) = \mathrm{tr}_{\#}(M, \omega, \omega_{\max})$. Let $\mathcal{E} \subseteq V(M) \times V(M)$ and $\mathcal{E}' \subseteq V(M') \times V(M')$ such that $(u, v) \in \mathcal{E}$ if and only if there are copies u', v' of u, v in M' with $(u', v') \in \mathcal{E}'$. Under this condition, $M + \mathcal{E}$ is connected if and only if $M' + \mathcal{E}'$ is connected since, by construction, vertices u, v of M are connected if and only if all copies of u and v in M' are connected and the connectedness relation is transitive. We go on to prove the claim of the lemma.
"$\Leftarrow$" Suppose that there is a function # such that $(M', \omega', \omega'_{\max})$ admits a solution $\mathcal{E}'$. Let $\mathcal{E}$ be the arc set that results from replacing each arc $(u', v') \in \mathcal{E}'$ by the corresponding arc (u, v) between the original vertices in M. By construction, the imbalance of a vertex v of M equals the sum of imbalances of its copies in M', implying that $M + \mathcal{E}$ is balanced. Furthermore, since $\mathcal{E}$ and $\mathcal{E}'$ fulfill the conditions of the observation in the first paragraph of this proof, we also know that $M + \mathcal{E}$ is connected. Finally, note that $\mathcal{E}$ has the same weight as $\mathcal{E}'$, implying that $\mathcal{E}$ is a solution for $(M, \omega, \omega_{\max})$.
"$\Rightarrow$" Let $(M, \omega, \omega_{\max})$ be a yes-instance of Weighted Multigraph Eulerian Extension with solution $\mathcal{E}$. For each connected component C of M, let k_C be the number of arcs in $\mathcal{E}$ that are outgoing of a vertex of C, that is, $k_C := |\mathcal{E} \cap (V(C) \times V(M))|$. Since $|\mathcal{E}| \leq k$, we have $\sum_{C \in \mathcal{C}_M} k_C \leq k$. Moreover, since $\mathcal{E}$ is an Eulerian extension for M, each k_C satisfies $k_C \geq |\mathcal{I}_C^+| = |\mathcal{I}_C|/2$. Thus, the function $\#(C) := k_C$ complies with (4.3). It remains to show that $(M', \omega', \omega'_{\max})$ is a yes-instance of Black/Gray Weighted Multigraph Eulerian Extension. To this end, we construct a solution $\mathcal{E}'$ for $(M', \omega', \omega'_{\max})$ as follows. For each path p from u to v in $\mathcal{E}$, we construct a path p' in $\mathcal{E}'$ by starting from a copy of u that is imbalanced in M'. Each time p reaches a vertex $x \neq v$, we select a copy of x whose black connected component is balanced in M' and has not yet been visited by $\mathcal{E}'$. Finally, we select a copy of v that is imbalanced in M' as terminal vertex of p'. Since there are k_C copies of each connected component of M in M', this selection is always possible. Since, by construction, the balance of a vertex v in M equals the sum of balances of its copies in M', we know that $M' + \mathcal{E}'$ is balanced. Furthermore, by the observation in the first paragraph of this proof, $M' + \mathcal{E}'$ is also connected. □

4.4.2 An Algorithm for the Weaker Variant

Having transformed an instance of Weighted Multigraph Eulerian Extension to an instance of Black/Gray Weighted Multigraph Eulerian Extension using the algorithm presented in Section 4.4.1, we can now exploit the simpler structure

of Black/Gray Weighted Multigraph Eulerian Extension in a dynamic programming algorithm. The main idea in this algorithm is to construct an Eulerian extension arc by arc while maintaining a set of connected components of the input multigraph that have already been visited.

Let $(M, \omega, \omega_{\max})$ be an instance of Black/Gray Weighted Multigraph Eulerian Extension and let C_M^{black} be the set of black connected components of M. For each subset $S \subseteq C_M^{\text{black}}$ and each pair of vertices $u, v \in V(S)$, our algorithm computes an entry $[S, u, v]$ of a three-dimensional dynamic programming table such that

$$[S, u, v] = \begin{array}{l}\text{minimum weight } \omega(\mathcal{E}) \text{ of an arc set } \mathcal{E} \text{ such that } \mathcal{E} + (v, u) \\ \text{is a black/gray Eulerian extension for } M[V(S)].\end{array} \tag{4.4}$$

If no black/gray Eulerian extension is possible with S, u, and v, then the entry $[S, u, v]$ is assigned "∞". The set S represents a submultigraph of M and the two vertices correspond to the endpoints of a (possibly "unfinished") path of an Eulerian extension (see Observation 4.4). The dynamic programming starts with computing the entries for sets S that contain exactly one connected component and augments each S step by step, finally computing the entries for $S = C_M^{\text{black}}$, which are used to derive a minimum-weight black/gray Eulerian extension for M with respect to ω. In the following, we describe the initialization process for the entries.

For each $C \in C_M^{\text{black}}$ not containing imbalanced vertices and for each $u, v \in V(C)$, set

$$[\{C\}, u, v] := \begin{cases} 0, & \text{if } u = v, \\ \infty, & \text{otherwise.} \end{cases}$$

This assignment is correct, that is, it satisfies (4.4) by setting $\mathcal{E} := \emptyset$ (which has obviously minimum weight) because adding an arc to a balanced component can only keep the component balanced if the added arc is a loop. Thus $\mathcal{E} + (v, u)$ is a black/gray Eulerian extension for $M[V(C)]$ since the only connected component has exactly one incoming arc as well as one outgoing arc in $\mathcal{E}$ (in this case, the incoming and outgoing arc is (v, u)).

For each $C \in C_M^{\text{black}}$ containing two imbalanced vertices $x \in \mathcal{I}_M^-$ and $y \in \mathcal{I}_M^+$, and each $u, v \in V(C)$, set

$$[\{C\}, u, v] := \begin{cases} 0, & \text{if } u = x \text{ and } v = y, \\ \infty, & \text{otherwise.} \end{cases}$$

This assignment satisfies (4.4) since, by definition of black/gray Eulerian extension, x and y are the only imbalanced vertices of C and both are balanced adding (y, x) (that is, by using $\mathcal{E} = \emptyset$). For the same reasons as above, $\mathcal{E} + (v, u)$ is a black/gray Eulerian extension for $M[V(C)]$.

Next, we describe the computation of the entries for larger sets S. We assume that all entries for sets S' with $|S'| < |S|$ have already been computed. For a given $S \subseteq C_M^{\text{black}}$ with $|S| > 1$ and vertices $u, v \in V(S)$, the entry $[S, u, v]$ is computed as follows. Let $C \in S$ denote the black component of M that contains v and let $S' := S \setminus \{C\}$. If C is balanced, then distinguish the following three subcases:

1. If $u = v$ and there is a gray arc between C and S', then set
$$[S, u, v] := \min_{u', v' \in V(S')} \{[S', u', v'] + \omega(v', u')\}.$$
2. If $u \in V(S')$, then set
$$[S, u, v] := \min_{w \in V(S')} \{[S', u, w] + \omega(w, v)\}.$$
3. Otherwise, set $[S, u, v] := \infty$.

If C contains two imbalanced vertices $x \in \mathcal{I}_M^-$ and $y \in \mathcal{I}_M^+$, then we distinguish the following three subcases:

1. If $u = x$, $v = y$, and there is a gray arc between C and S', then set
$$[S, u, v] := \min_{u', v' \in V(S')} \{[S', u', v'] + \omega(v', u')\}.$$
2. If $u \in V(S')$ and $v = y$, then set
$$[S, u, v] := \min_{w \in V(S')} \{[S', u, w] + \omega(w, x)\}.$$
3. Otherwise, set $[S, u, v] := \infty$.

Finally, the weight ω_{opt} of an optimal black/gray Eulerian extension for (M', ω) is computed as follows:

$$\omega_{\text{opt}} := \min_{u, v \in V(C_M^{\text{black}})} \{[C_M^{\text{black}}, u, v] + \omega(v, u)\}.$$

This follows immediately from (4.4). A corresponding black/gray Eulerian extension can be computed by storing each solution $\mathcal{E}$ in addition to its weight in each entry of the dynamic programming table.

Lemma 4.5. Black/Gray Weighted Multigraph Eulerian Extension *can be solved in* $O(2^k \cdot n^3)$ *time, where k denotes the size of the solution.*

Proof. In the above description of the dynamic programming algorithm, we already established correctness for $|S| = 1$, that is, the case that S contains a single black component. In the following, consider $|S| > 1$. We prove that the

semantics of $[S, u, v]$ (see (4.4)) is met by the computation. To this end, recall that C denotes the black component of v in M and $S' := S \setminus \{C\}$.

The correctness proofs for the case that C is balanced and the case that C is imbalanced are very similar. Hence, we merge the correctness arguments for both cases in each of the three subcases 1–3 of the algorithm above. For ease of presentation, let $\mathcal{E}(S, u, v)$ be an arc set that corresponds to the entry $[S, u, v]$ (that is, $[S, u, v]$ contains the total weight of $\mathcal{E}(S, u, v)$) as defined in (4.4).

Subcase 1: In Subcase 1, we assume that u and v are in the same connected component $C \in S$. Being a black/gray Eulerian extension for $M[V(S)]$, $\mathcal{E}(S, u, v)+(v, u)$ contains only one arc incoming to and one arc outgoing of C. Since both u and v are in C, no arc in $\mathcal{E}(S, u, v)$ starts or ends in C. Hence, C is connected to $M[V(S')]$ by gray arcs. If C is balanced, then (4.4) is satisfied only if $u = v$, as otherwise inserting (v, u) would make v imbalanced. If C contains two imbalanced vertices $x \in \mathcal{I}^-_M$ and $y \in \mathcal{I}^+_M$, then (4.4) is satisfied only if $u = x$ and $v = y$, as otherwise inserting (v, u) does not balance u and v. Thus, according to (4.4), we only have to ensure that $\mathcal{E}(S, u, v)$ is a black/gray Eulerian extension for $M[V(S')]$. To this end, we try all possible $\mathcal{E}(S', u', v')$ and add the arc (v', u'). The resulting arc set of minimum weight is assigned to $\mathcal{E}(S, u, v)$. Note that entries $[S', u', v']$ with $u' = v'$ do not have to be considered, since the arc set $\mathcal{E}(S', u', v') + (v', u') + (v, u)$ would contain the loop (u', v').

Subcase 2: In Subcase 2, we assume that the connected component C is connected to $M[V(S')]$ by an arc $a \in \mathcal{E}(S, u, v)$. By an argument of Subcase 1, $u \in V(C)$ contradicts the existence of black arcs between C and $M[V(S')]$. Hence, $u \in V(S')$. Furthermore, there cannot be an arc from C to $M[V(S')]$ in $\mathcal{E}(S, u, v)$, since together with the arc (v, u) there would be two arcs that are outgoing of C. Hence, we just have to guess the arc from $M[V(S')]$ to C in $\mathcal{E}(S, u, v)$. If C is balanced, then we try all possible $\mathcal{E}(S', u, w)$ and add the arc (w, v). The resulting arc set of minimum weight is assigned to $\mathcal{E}(S, u, v)$. If C is imbalanced, that is, it contains two imbalanced vertices $x \in \mathcal{I}^-_M$ and $y \in \mathcal{I}^+_M$, then we try all possible $\mathcal{E}(S', u, w)$ and add the arc (w, x). The resulting arc set of minimum weight is assigned to $\mathcal{E}(S, u, v)$. Observe that this is the only way for $\mathcal{E}(S, u, v)$ to balance x and y, which is necessary to satisfy (4.4).

Subcase 3: In Subcase 3, $[S, u, v]$ does not correspond to an arc set $\mathcal{E}(S, u, v)$ such that $\mathcal{E}(S, u, v) + (v, u)$ is a black/gray Eulerian extension for $M[V(S)]$; therefore, we set $[S, u, v] := \infty$. This concludes the correctness proof of the updating process for $[S, u, v]$.

To finish the proof of Lemma 4.5, we show the running time of the dynamic programming. Clearly, if S contains exactly one connected component, then $[S, u, v]$ can be computed in constant time. Otherwise, since $S \subseteq \mathcal{C}^{\text{black}}_M$, $u, v \in V(S)$, and $|\mathcal{C}^{\text{black}}_M| \leq k$, there are $O(2^k \cdot n)$ entries of the form $[S, u, u]$, each of which can

be computed in $O(n^2)$ time (Subcase 1). Furthermore, there are $O(2^k \cdot n^2)$ entries of the form $[S, u, v]$, each of which can be computed in $O(n)$ time (Subcase 2). We arrive at a total running time of $O(2^k \cdot n^3)$. □

4.4.3 The Complete Algorithm

An important part of the algorithm for Black/Gray Weighted Multigraph Eulerian Extension presented in Section 4.4.2 is to try all feasible functions # (recall that # assigns each connected component the number of times optimal solution visits this connected component; see Section 4.4.1). Since there are at most n connected components in the input, there are at most n^k such functions. Recall that k denotes the cardinality of the sought Eulerian extension (see Section 4.2 on page 115).

Observation 4.5. *There are at most n^k possible functions $\# : C_M \to \mathbb{N}$ as defined in (4.3) (on page 126), that is, there are at most n^k possible ways to replace each connected component of M by several copies such that there are at most k copies in total.*

The complete algorithm to solve Weighted Multigraph Eulerian Extension performs two steps:

Step 1: Compute an arbitrary component-respecting bijection $\mu : \mathcal{I}_M^- \to \mathcal{I}_M^+$ in $O(m^2)$ time.

Step 2: For all n^k possible functions $\# : C_M \to \mathbb{N} \setminus \{0\}$ (see Observation 4.5), transform the instance and solve the transformed instance of Black/Gray Weighted Multigraph Eulerian Extension in $O(2^k \cdot n^3)$ time (see Lemma 4.5).

The correctness of this algorithm follows directly from the correctness of the transformation algorithm (see Lemma 4.4) and Lemma 4.5. The running time is $O(n^k 2^k \cdot n^4)$.

The presented algorithm depends heavily on the knowledge of the parameter k. Since we do not know k in advance, we run Step 1 and Step 2 for increasing values of k. Using geometric progression, we estimate

$$\sum_{i=0}^{k} O(n^i 2^i \cdot n^3) = O(n^3 \cdot \sum_{i=0}^{k} (2n)^i) = O(n^k 2^k \cdot n^4). \tag{4.5}$$

Theorem 4.3. Weighted Multigraph Eulerian Extension *can be solved in $O(n^k 2^k \cdot n^4)$ time.*

4.5 Adding an Intuitive Preprocessing

In the following, we describe a simple and intuitive preprocessing routine for Weighted Multigraph Eulerian Extension running in $O(n^3)$ time. It will produce equivalent instances that, due to their simplicity, do not require as much "guesswork" as more complex instances. Hence, the algorithm presented in Section 4.4 can be adapted to run in $O(4^k \cdot n^3)$ time, where c denotes the number of connected components in the input graph that contain arcs (equivalently, the number of connected components induced by the required arc in the corresponding Rural Postman instance). To this end, we replace the weight of a vertex pair (u, v) of the input multigraph M with the weight of the cheapest u-v-path of non-arcs in M. Having modified the weights between all vertices in this way, we can remove all isolated vertices from the multigraph. Thus, we avoid considering isolated vertices as connected components, which dramatically decreases the number of functions #, we have to consider.

Reduction Rule 4.1. *Let $(M, \omega, \omega_{max})$ be an instance of* Weighted Multigraph Eulerian Extension. *For each pair (u, v) of vertices of M, let $\omega_\Delta((u, v))$ denote the length of a shortest path from u to v in M according to ω and let V_I denote the set of isolated vertices in M. Then, return the instance $(M - V_I, \omega_\Delta, \omega_{max})$.*

Reduction Rule 4.1 can be applied by computing all-pairs shortest paths using the well-known Floyd-Warshall algorithm running in $O(n^3)$ time. Also note that this operation imposes the triangle inequality on the weight-function ω.

Observation 4.6. *Reduction Rule 4.1 is correct and can be applied exhaustively in $O(n^3)$ time. In an instance $(M, \omega, \omega_{max})$ that is reduced with respect to Reduction Rule 4.1, the weight function ω respects the triangle inequality, that is, for each three vertices $u, v, w \in V(M)$, it holds that*

$$\omega(u, v) + \omega(v, w) \geq \omega(u, w).$$

Proof of correctness (sketch). We show that the original instance is a yes-instance if and only if the new instance is a yes-instance.
"⇒": Consider an optimal solution $\mathcal{E}$ for the old instance. Clearly, no path of arcs in $\mathcal{E}$ starts or ends in an isolated vertex. Hence, replacing each subpath whose inner vertices are in V_I by a direct connection does not increase the weight of $\mathcal{E}$ measured by ω_Δ. Let $\mathcal{E}'$ denote the result of this replacement. Since $\omega_\Delta(u, v) \leq \omega(u, v)$ for each pair of vertices $u, v \in V \setminus V_I$, $\mathcal{E}'$ is a solution for the new instance.
"⇐": To obtain a solution for the old instance from a solution $\mathcal{E}'$ for the new instance, we can just replace each arc $(u, v) \in \mathcal{E}'$ by the path used to compute $\omega_\Delta(u, v)$ in M. □

Input instances $(M, \omega, \omega_{\max})$ that are free of isolated vertices and whose weight function respects the triangle inequality exhibit much less complex extension paths then the ones considered in Section 4.4. We capture this simplicity in the following two lemmas.
First, the fact that each connected component of M contains an arc implies that only functions # (as defined in (4.3)) that include at least one copy per connected component of M are feasible. Therefore, we can show a better bound on the number of such functions.

Lemma 4.6. *For a multigraph M that does not contain isolated vertices, there are at most 2^k possible ways to replace each connected component of M by several copies such that there are at most k copies in total, that is, there are at most 2^k possible functions $\# : C_M \to \mathbb{N}$ as defined in (4.3).*

Proof. Let $c := |C_M|$. Since M does not contain isolated vertices, each connected component of M contributes at least one copy and it remains to distribute at most $k - c$ copies to c components. This is equal to choosing at most $k - c$ from c elements with repetition. There are at most $\sum_{i \leq k-c} \binom{c+i-1}{i} \leq 2^k$ possible ways to do so. □

The running time computed in Section 4.4.3 (Equation 4.5) then changes to

$$\sum_{i=0}^{k} \mathrm{O}(2^i 2^i \cdot n^3) = \mathrm{O}(n^3 \cdot \sum_{i=0}^{k} 4^i) = \mathrm{O}(4^k \cdot n^3).$$

Next, we bound the range of values that are suitable for k.

Lemma 4.7. *Let $(M, \omega, \omega_{max})$ be an instance of* Weighted Multigraph Eulerian Extension *that is reduced with respect to Reduction Rule 4.1. Then, there is an optimal Eulerian extension $\mathcal{E}$ for M such that $|\mathcal{E}| \leq |\mathcal{I}_M^+| + 2|C_M|$.*

Proof. Consider an optimal Eulerian extension $\mathcal{E}$ for M that cannot be shortcut, that is, for each extension path p of $\mathcal{E}$, replacing a subpath (u, v, w) of p by (u, w) does not yield an Eulerian extension for M. Since M is reduced with respect to Reduction Rule 4.1, Observation 4.6 implies that a shortcut Eulerian extension would not suffer an increased weight and, hence, we could simply shortcut a given Eulerian extension until any shortcutting does not yield an Eulerian extension. Thus, $\mathcal{E}$ exists. Let χ_1 and χ_2 denote the number of extension paths in $\mathcal{E}$ whose length is exactly one and at least two, respectively.
First, consider an extension path p of length one in $\mathcal{E}$. Then, both its endpoints are imbalanced in M. Thus, adding p to M decreases the imbalance of M and, hence,

$$\chi_1 \leq |\mathcal{I}_M^+|. \tag{4.6}$$

Next, consider an extension path p of length at least two in $\mathcal{E}$ and let (u, v) and (v, w) be two consecutive arcs of p. Clearly, replacing the subpath (u, v, w) by (u, w) in p does not change the balance of a vertex in $M + \mathcal{E}$. However, since $\mathcal{E}$ cannot be shortcut, replacing the subpath disconnects $M + \mathcal{E}$. Thus, each two consecutive arcs of p decrease the number of connected component. Hence, denoting the length of the i'th extension path of length at least two by ℓ_i, it holds that

$$\sum_{i=1}^{\chi_2} (\ell_i - 1) \leq |C_M|. \tag{4.7}$$

Since $\ell_i \geq 2$, (4.7) also implies

$$\chi_2 \leq |C_M|. \tag{4.8}$$

Then, we can bound

$$|\mathcal{E}| = \chi_1 + \sum_{i=1}^{\chi_2} \ell_i = \chi_1 + \chi_2 + \sum_{i=1}^{\chi_2} (\ell_i - 1) \overset{(4.7)}{\leq} \chi_1 + \chi_2 + |C_M| \overset{(4.6),(4.8)}{\leq} |\mathcal{I}_M^+| + 2|C_M|.$$

□

Plugging Lemma 4.6 and Lemma 4.7 into the algorithm summed up in Section 4.4.3, we arrive at the main theorem.

Theorem 4.4. Weighted Multigraph Eulerian Extension *on multigraphs M can be solved in $O(4^k \cdot n^3) \subseteq O(4^{b+2c} \cdot n^3)$ time, where $b = |\mathcal{I}_M^+|$ denotes the sum of positive balances in M, $c = |C_M|$ denotes the number of connected component in M, and k denotes the cardinality of the sought solution.*

As a consequence of the characterization provided in Proposition 4.1, we can analogously solve Rural Postman parameterized by k, denoting the number of non-required arcs in the sought solution.

Corollary 4.3. Rural Postman *can be solved in $O(4^k \cdot n^3)$ time.*

Finally, note that a solution for the original instance can be reconstructed from a solution for the preprocessed instance. To this end, simply replace each arc by the shortest path between its endpoints in the original instance.

4.6 Lower bounds for Kernelization

As we have seen in Section 4.5, preprocessing supports the development of faster algorithms for Weighted Multigraph Eulerian Extension by imposing

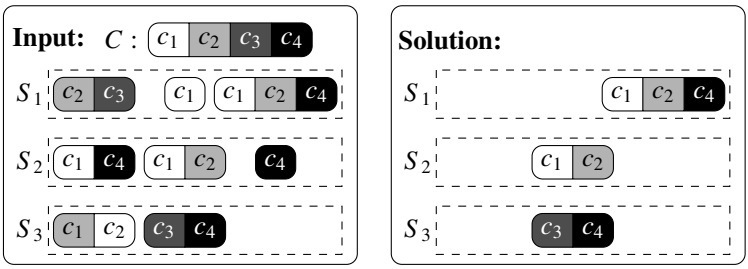
restrictions on the input that can be exploited in both design and analysis. In this section, we complement this result by demonstrating that, under reasonable complexity-theoretic assumptions, Weighted Multigraph Eulerian Extension does not admit a polynomial-size kernel with respect to any combination of

- the number of connected components c of the input,
- the imbalance b of the input, and
- the smallest size k of a minimum-weight Eulerian extension for the input.

It is sufficient to show the claim for k since both b and c are "stronger" parameters [134], that is, $b \leq k$ and $c \leq k$ hold for any input multigraph. In the following, we sketch a proof showing that Weighted Multigraph Eulerian Extension does not admit a polynomial-size problem kernel unless NP $\subseteq$ coNP/ poly. To this end, recall the notion of cross compositions (Section 1.2.4 on page 18).

In the following, we first sketch that a problem called Switch Set Cover does not admit a polynomial-size problem kernel and then sketch a polynomial-time polynomial-parameter reduction (see Theorem 1.2.4, page 20) of Switch Set Cover to Weighted Multigraph Eulerian Extension. In fact, all instances produced by this reduction are also instances of Two-Dimensional Eulerian Extension (see Section 4.1, page 111), implying that not even this restricted version of Weighted Multigraph Eulerian Extension admits a polynomial-size problem kernel with respect to the parameter k. Intuitively, in Switch Set Cover, we have multiple opportunities to choose one of a number of sets called *positions*. Each such opportunity is called *switch* and the goal is to hit all elements with the chosen positions. Note that, for each switch, exactly one position must be chosen.

Switch Set Cover
Input: A set C (of colors) and lists S_i (switches) of subsets of C (positions).
Question: Can we choose a position for each switch such that each color in C is contained in at least one of the chosen positions?



Switch Set Cover is NP-complete since it contains the well-known Set Cover problem as the special case that all switches are equal (for details, refer to [178]). We consider Switch Set Cover parameterized by the combined parameter "number c of colors" plus "number s of switches", that is $c + s = |C| + |S|$, where S denotes the set of lists S_i in the input. Then, a rather involved cross composition (see Definition 1.6 on page 18) for Switch Set Cover with respect to $c + s$ and a polynomial-parameter polynomial-time reduction shows that Two-Dimensional Eulerian Extension does not admit a polynomial-size problem kernel with respect to k unless NP $\subseteq$ coNP/ poly. In the following, we sketch this cross composition. To this end, assume that t instances of Switch Set Cover are given, each on the same color set C with and each with the same number s of switches. We denote the switches of instance I_i, $1 \leq i \leq t$, by S^i_β, $1 \leq \beta \leq s$.

Step 1: For each $1 \leq \alpha \leq \log t$ and $1 \leq \beta \leq s$ introduce two new colors 0^α_β and 1^α_β and, to each switch S^i_β of an instance I_i, add 0^α_β or 1^α_β if the α'th bit[32] of i is 0 or 1, respectively.

Step 2: For each $1 \leq \beta \leq s$, merge S^i_β for all $1 \leq i \leq t$ into one switch S^*_β.

Step 3: For each $1 \leq \alpha \leq \log t$, create a new switch S'_α containing the two positions $\{0^\alpha_\beta : 1 \leq \beta \leq s\}$ and $\{1^\alpha_\beta : 1 \leq \beta \leq s\}$.

The resulting instance of Switch Set Cover contains $c + 2s \log t$ colors and $s + \log t$ switches. It can be shown to be a yes-instance if and only if one of the instances I_i is a yes-instance (see [178]). Hence, the presented algorithm constitutes a cross composition [29] of Switch Set Cover, implying that the existence of a polynomial-size problem kernel for SSC with respect to the parameter $c + s$ is unlikely.

The next step is to provide a polynomial-time polynomial-parameter reduction from Switch Set Cover to Two-Dimensional Eulerian Extension. Intuitively speaking, each color c_i of the input instance can be modeled as a cycle of arcs and each switch can be seen as a two-dimensional box. Now, since in Two-Dimensional Eulerian Extension only arcs going from (x_1, y_1) to (x_2, y_2) with $x_1 \leq x_2$ and $y_1 \leq y_2$ can be inserted, the boxes can be placed in a diagonal such that no arc between two boxes can be inserted. Finally, vertices of the cycles representing the colors are placed into each box according to the position in each switch. Like the boxes, positions can be placed in a diagonal inside a box such that no arc can be inserted between them. To argue that the parameter k of the constructed instance is polynomially bounded in $c + s$, assume that each position contains at most c colors. By forcing a solution to contain at most one path per box, we ensure $k \in O(c \cdot s)$. This leads to the following theorem.

Theorem 4.5. Two-Dimensional Eulerian Extension *(and thus,* Weighted Multi-

[32] The α'th bit of a number i is $\lfloor i/2^\alpha \rfloor \bmod 2$.

GRAPH EULERIAN EXTENSION*) does not admit a polynomial-size problem kernel with respect to the parameter "number k of extension arcs" unless* co NP $\subseteq$ NP/poly.

4.7 Conclusion

We focused on Eulerian extension problems (and, due to parameterized equivalence, the RURAL POSTMAN problem), leaving the "editing version", where adding *and* deleting arcs is allowed, yet unstudied. We showed how, although WEIGHTED MULTIGRAPH EULERIAN EXTENSION resists kernelization, preprocessing helps to show fixed-parameter tractability of this vital arc routing problem [67].

Eulerian extension problems still offer a rich field of challenges for future research in terms of multivariate algorithmics [82, 161]. More specifically, we concentrated on the parameterized complexity with respect to the parameter "number of extension arcs", but there are many natural structural parameters that make sense. For instance, it would be particularly interesting to determine the parameterized complexity with respect to the parameter "number of weakly connected components" in a WEIGHTED MULTIGRAPH EULERIAN EXTENSION instance. In this context, Orloff [164] observed that "the determining factor in the complexity of the problem seems to be the number c of connected components in the required edge set"; Frederickson [96, 97] noted "the existence of an exact recursive algorithm that is exponential only in the number of disconnected components." However, this statement refers to an $n^{O(c)}$-time algorithm, leaving open whether the problem is fixed-parameter tractable with respect to the parameter c. It is not even apparent whether the problem is in $W[1]$.

The lower bound result for WEIGHTED MULTIGRAPH EULERIAN EXTENSION with respect to the parameter c that we presented in Section 4.6 motivates opening up two new lines of research. First, weaker parameters or parameter combinations might allow the development of polynomial-size problem kernels for WEIGHTED MULTIGRAPH EULERIAN EXTENSION. A second way around the shown lower bounds could be to consider a more general concept of kernelization, as presented in Chapter 5.

Since the real-world applications of RURAL POSTMAN center around street networks, it may be worth considering the problem on planar graphs or graphs at small distance to planar graphs (for example, bounded genus). Here, "protrusion-based" kernelization methods [27, 132] or the subexponential-time framework of Demaine et al. [59] may be applicable.

For further future work, we also consider the study of the currently unexplored undirected and non-multigraph versions of WEIGHTED MULTIGRAPH EULERIAN EXTENSION interesting. We conjecture that similar algorithmic approaches may allow for similar results.

Chapter 5

Between Turing and Classical Kernelization

As we have demonstrated in the previous chapters, preprocessing in general, and kernelization in particular, is a very important tool in designing fast algorithms for NP-hard problems. However, as we have also seen, polynomial-size problem kernels are unlikely to exist for some parameterized problems (for example, tw-Vertex Cover or k-Weighted Multigraph Eulerian Extension). In the recent past, methods have been developed to deal with this "incompressibility". We discuss some of these methods and consider a promising way to handle hardness of kernelization: the so-called "Turing kernel" and variants thereof. We hope that considering "weaker" variants of Turing kernelization helps in proving meta results for this very young concept. We will argue that, although Turing kernelization and its variants are somewhat underused, their practical relevance rivals that of classical kernelization. This warrants further considerations in the future. Thus, our work can be seen as a stepping stone in this direction.

5.1 Introduction

Almost as old as the theorem that each fixed-parameter tractable problem has a problem kernel [39] is the question whether all fixed-parameter tractable problems have *polynomial-size* problem kernels. This question was long standing and has only recently been answered [26, 95], if one is willing to assume NP $\not\subseteq$ coNP/poly. Since NP $\subseteq$ co NP/poly implies a collapse of the polynomial hierarchy to its *third* level [26, 95], this is a weaker conjecture than P $\neq$ NP (which implies a total collapse of the polynomial hierarchy). Still, even a collapse to the third level is widely unexpected. As in classical complexity theory, lower-bound results are not a reason to despair but rather a motivator to develop ways around these barriers. In the case of polynomial-time preprocessing, there are two immediate ways around lower bounds: we can either try to find weaker parameters or extend the boundaries of classical kernelization. As the former was discussed in Chapter 2, we will consider the latter here.

More permissive kernelization concepts. In the recent past, kernelization lower bounds motivated formalization of some forms of preprocessing that are more permissive than classical kernelization. In the following, we give examples for the most prominent ones.

Partial kernelization. Although producing an instance whose *size* is bounded polynomially in the parameter, it makes sense to consider some other measure of difficulty. For instance, Betzler et al. [17] developed polynomial-time algorithms that shrink one *dimension* of some two-dimensional problems. They present a polynomial-time algorithm that, given an instance of the Kemeny Score problem, produces an equivalent instance whose number of candidates is bounded polynomially in the considered parameter.

Turing kernelization. Also called "cheat kernel" [25], this form of kernelization allows the output of multiple instances instead of just one [88, 143]. In this sense, it is very similar to the notion of "Turing reductions" [57, 169, 190] (or "Cook reductions" [52]). Lokshtanov [143] defines Turing kernelizations as polynomial-time algorithms that have access to an oracle which accepts only inputs whose size is polynomial in the parameter.

α-fidelity kernelization. A way of dealing with hard problems is to give up the search for optimal solutions and accept approximate solutions. If the task of computing a polynomial-size problem kernel is hard, then one may be willing to accept an approximate problem kernel. More formally, for the parameterized decision version k-Q of a minimization problem and a function $g : \mathbb{N} \to \mathbb{N}$, Fellows et al. [85] define an *α-fidelity kernelization* as a polynomial-time algorithm that, given an instance $((x, t), k)$ of k-Q outputs an instance $((x', t'), k')$ of k-Q such that

1. $(x, t) \in Q \Rightarrow (x', t') \in Q$,
2. $(x', t') \in Q \Rightarrow (x, \alpha \cdot t) \in Q$,
3. $|x'| \leq g(k)$, and
4. $k' \leq k$.

Herein g is the size of the α-fidelity kernel.

Weak kernelization. A popular technique for developing fixed-parameter algorithms is "brute force on the problem kernel", that is, first computing a polynomial-size problem kernel and then running a brute-force algorithm on the problem kernel. To allow this, much less strict definitions of "problem kernel" suffice. Jiang and Zhu [127] give such a definition. They call a polynomial-time algorithm that bounds the "search space" (the space of possible solutions) of a given instance *weak kernel* [127, 128] and give an example for the Sorting with Minimum Unsigned Reversals problem.

In this chapter, we focus on Turing kernelizations for the following reasons.
First, most problems we consider are NP-complete and so, algorithms that solve these problems typically have exponential running time. Hence, for most parameterized problems k-Q, it is faster to solve poly(n) instances of size $g(k)$ than solving just one instance of size n. An extreme example is the heuristic reduction rule employed by almost all implementation of branching algorithms on graphs: If the graph is disconnected, run the algorithm on each connected component separately and combine the solutions. This can be seen as Turing kernelization with respect to the parameter "size n_c of a largest connected component". For an exemplary $O(2^n \cdot n)$-time algorithm, this Turing kernel improves the running time to $O(2^{n_c} \cdot n^2)$.
Second, altough Turing kernelization does not offer parallel processing of its queries, the variant we will consider behaves well in this regard. Hence, with enough machines, the disadvantage of having to compute multiple instances instead of just one disappears completely. Keeping an eye on modern developments in computer hardware as well as so-called "cloud-computing" and "crowd-computing" possibilities, the availability of massively parallel computation facilities does not seem far-fetched.
Third, Turing kernelizations may even be preferable to classical kernelization: since Turing kernelizations are more permissive in the number of created instances, it may be possible to achieve smaller instance-sizes than produced by classical kernelizations. For example, a Turing kernelization that computes $O(n)$ instances of size $O(k)$ may well be preferable to a classical kernelization that computes one instance of size $O(k^2)$. In the spirit of efficient kernelization (see Chapter 3), this train of thought carries over to the running time of the kernelization. Both arguments are reinforced by the prior discussion about parallel processing.

Previous work on Turing kernelization. In the following, we define Turing kernelization and give examples from the literature. To formally define Turing kernelization, we need the notion of "oracles". For a set X, consider a machine that can answer $x \in X$ for any x in constant time. We call this machine *oracle* for X.

Definition 5.1. *Let k-Q be a parameterized problem and let $g, g' : \mathbb{N} \to \mathbb{N}$ be computable. An algorithm that, for each instance (x, k), decides $(x, k) \in k$-Q in polynomial time using an oracle for $\{(x', k') : |x'| \leq g(k) \wedge k' \leq g'(k) \wedge (x', k') \in k\text{-}Q\}$ is called a* Turing kernelization *for k-Q. The sequence of queries posed to the oracle is called* Turing kernel. *We call $g(k)$ the* size *of the Turing kernel.*

Turing kernelizations defy the lower-bound framework for classical kernelization [26, 95]. Dell and van Melkebeek [58] recently presented a slightly different

approach to showing kernelization hardness, which incorporates so-called "oracle communication protocols". This framework is a good candidate for a lower bound framework for Turing kernelizations; however, no result in this direction is known to date. Instead, Hermelin et al. [117] established a hierarchy of kernelization hardness, much like the W-hierarchy in parameterized complexity. They conjecture that k-Connected Vertex Cover and $(k \log n)$-Clique, where n refers to the number of vertices in the input instance, do not admit polynomial-size Turing kernels. So far, only few polynomial-size Turing kernelizations for parameterized problems that do not admit polynomial-size classical kernels are known.

1. The k-Leaf Out-Branching and k-Leaf Out-Tree problems ask for a connected subdigraph T of an input digraph G such that the underlying undirected graph of T is a (spanning) tree and there is a dedicated "root" in T from which all arcs of T are ordered away and T has at least k sinks. Fernau et al. [88] showed that these problems admit polynomial-size Turing kernels with respect to k by proving that their "rooted variants" (when a specific vertex of G is chosen as root) admit polynomial-size problem kernels, while the unrooted problems were shown to be compositional and, therefore, do not admit polynomial-size problem kernels (see Section 1.2.4).
2. Given a graph G, the s-Club problem asks for a subgraph G' of G that contains exactly a given number k of vertices and has diameter at most s. Schäfer et al. [172] showed that s-Club does not admit a polynomial-size problem kernel with respect to the parameter k. However, they developed a Turing kernel posing n queries each containing $O(k^3)$ vertices (corresponding to an n-query $O(k^6)$-size Turing kernel).
3. A rather pathological, but easy to see example is the Δ-Clique problem, that is, Clique parameterized by the maximum degree Δ of the input graph G. As discussed by Hermelin et al. [117], an algorithm that, given an instance (G, k) of Clique poses the query $(G[N[v]], k)$ for each vertex v in G and returns "yes" if and only if one of the queries returns "yes" constitutes an n-query $\binom{\Delta+1}{2}$-size Turing kernel.

It is common to these examples that the full power of the Turing kernelization concept is not used. Instead, the input instance is simply mapped to a set of queries whose answer then determines whether the input was a yes-instance or not. Turing kernelizations, however, are allowed to compute different instances, based on oracle answers to previous queries. It is reasonable to ask whether this unused power can be harnessed to allow stating better bounds in size or running time of Turing kernels. As we will see, this is indeed the case for Turing kernels with poly(k) queries. Motivated by this result, we initiate the search for a problem that allows for a polynomial-size Turing kernel with poly(k) queries but no polynomial-size classical kernel. We show that, even if such a problem

exists, we cannot use composition-based lower bounds to prove that it does not admit a polynomial-size classical kernelization.
Further investigation into the possibilities of trading properties of Turing kernelization, we exemplary show that a reasonable parameterization of Clique allows improving the size of the posed queries at a cost of quadratically increased number of queries.

5.2 Preliminaries

Let Q be a problem. The *characteristic function* χ_Q of Q maps yes-instances of Q to true and no-instances of Q to false. For an integer $m \in \mathbb{N}^+$, the m-ary characteristic function χ_Q^m of Q maps all vectors of instances $(I_0, I_1, \ldots, I_{m-1})$ of Q to $(\chi_Q(I_0), \chi_Q(I_1), \ldots, \chi_Q(I_{m-1}))$. If clear from context, we may simply write χ_Q instead of χ_Q^m. Recall that we use k-Q as an abbreviation for "Q parameterized by k", where k is to be understood as a letter representing a variable, not a variable representing a value (see page 13).
Let $m \in \mathbb{N}$ and let $f : \{\text{false}, \text{true}\}^m \to \{\text{false}, \text{true}\}$ be an m-ary boolean function and let φ be a boolean formula on m variables $x_0, x_1, \ldots, x_{m-1}$. A vector in $\{\text{false}, \text{true}\}^m$ represents an assignment of variables of φ to true or false in the natural way. We say that φ *represents* f if, for all $z \in \{\text{false}, \text{true}\}^m$, the formula φ evaluates to $f(z)$ under the assignment z. In this spirit, we may write $\varphi(z)$ instead of $f(z)$. Furthermore, the *size* $|\varphi|$ of φ is the length of a shortest bit-representation of φ.
We say a kernelization algorithm that poses m queries, each of size s, is an m-query s-size kernelization. For a vector v, the term v_i refers to its i'th element.
In this section, we use the concept of nondeterminism (see for example [10, 165]). Intuitively, a *nondeterministic algorithm* is an algorithm that may perform a "nondeterministic guess" step. In this step, a new boolean variable x is introduced and the computation branches into two independent paths that start in the same state, except for the new variable x which is false in one path and true in the other. Each of these paths can, at any point, again perform a "nondeterministic guess". The running time of such an algorithm is the maximum of the running times of the individual paths.

5.3 Introducing Truthtable Kernelization

Recall the definitions of Turing and classical kernelization and observe that both concepts strongly resemble similar concepts in recursion theory. In fact, the classical kernelization is a polynomial-time many-one (or Karp) reduction

of a parameterized problem k-Q to itself, such that all instances created by the reduction are bounded by a function in the parameter. Likewise, a Turing kernelization is a Turing reduction of a parameterized problem k-Q to itself, such that the size of each query is bounded by a function in the parameter. It is natural to consider other reduction concepts that have been developed in recursion theory. One of these concepts is the *truthtable* reduction, that allows creation of a set of instances (queries) and a boolean formula with these queries as variables such that the input instance is a yes-instance if and only if if the formula evaluates to true under the assignment implied by the answers to the posed queries. Formally, we define the following.

Definition 5.2. *Let k-Q be a parameterized problem and let $g, g' : \mathbb{N} \to \mathbb{N}$ be computable functions. A polynomial-time algorithm that, given an instance (x, k) of k-Q, produces*

1. *a vector of m instances $\mathcal{I}_x := (x_i, k_i)$, $0 \leq i < m$ of k-Q and*
2. *a boolean formula* eval_x *on m variables*

such that

(i) $(x, k) \in k\text{-}Q \Leftrightarrow \text{eval}_x(\chi_{k\text{-}Q}(\mathcal{I}_x)) = \text{true}$,
(ii) $\forall_{(x',k')\in\mathcal{I}} |x| \leq g(k)$, *and*
(iii) $\forall_{(x',k')\in\mathcal{I}} k' \leq g'(k)$

is called a truthtable kernelization *for k-Q. The pair $(\mathcal{I}_x, \text{eval}_x)$ is called a* truthtable kernel *for k-Q. We call $g(k)$ the* size *of the truthtable kernel and* eval_x *the* evaluation *formula. The function $g''(x, k) = |\text{eval}_x|$ is called the* size *of the evaluation formula and the truthtable kernelization is called $g''(x, k)$-*evaluable.

Observe that all Turing kernelizations mentioned in Section 5.1 can be stated as polynomial-size truthtable kernelizations with $O(n)$ queries and disjunction evaluation formula (which we call $\text{TT}^{\text{Dis}}_{O(n)}$).
In this chapter, we consider certain restrictions of Turing and truthtable kernelizations. Most of these restrictions limit the size of the evaluation formula. Other kinds of restrictions that we apply to truthtable kernelization are the structure of eval_x and the number and size of queries posed by the algorithm.

Definition 5.3. *The set of parameterized problems k-Q that admit a* poly(k)*-size...*

... classical (many-one) kernel is denoted by DF.[33]
... m-query Turing kernel is denoted by Tu_m.
... m-query truthtable kernel is denoted by TT_m.

[33] This nomenclature is rooted in the historical backround of "classical" kernelization, which was introduced by Downey and Fellows [66].

... m-query truthtable kernel such that for all instances (x, k) of k-Q, the evaluation formula eval_x *has type X is denoted by* TT_m^{X} *(for example,* TT_m^{CNF} *for CNF-formulas,* TT_m^{Con} *for conjunctions, and* TT_m^{Dis} *for disjunctions).*

... m-query truthtable kernel such that the size of the evaluation formula is bounded by a function $g(n, k)$ is denoted by $\text{TT}_m^{g(n,k)}$ *(for example,* $\text{TT}_m^{\text{O}(n)}$ *and* $\text{TT}_m^{\text{poly}(k)}$*).*

In contrast to Turing kernelizations, the queries posed by a truthtable kernelization do not depend on answers to previous queries. Thus, all queries can be computed in a first phase and, in a second phase, all queries can be worked on in parallel. With today's crowdsourcing and cloudsourcing possibilities, the growing trend to include more cores in a CPU, and the massive parallelism offered by programming GPUs, parallel algorithms are desirable to get results more quickly. This makes truthtable kernelizations more intersting than Turing kernelizations, even for problems whose solution algorithms do not benefit from parallel execution possibilities.

In this chapter, we make initial observations and state first theorems on relations between different kinds of kernelizations defined in Definition 5.3. In particular, we prove in Section 5.3.1 that poly(k)-size poly(k)-evaluable truthtable kernelizations with m queries can be turned into poly(k)-size Turing kernels posing only $\log m$ queries and vice versa. This means that trading parallelism for queries is possible for the class of problems admitting poly(k)-size poly(k)-evaluable truthtable kernelizations. We initiate the search for such problems that, additionally, do not admit polynomial-size classical kernelizations. We show that the truthtable kernelizations admitted by these problems have properties that set them aside from all known truthtable kernelizations. More importantly, we show in Section 5.3.2 that we cannot use compositions to tell $\text{TT}_{\text{poly}(k)}^{\text{poly}(k)}$ and DF apart.

Finally, in Section 5.3.3, we show that trading the number of queries for their size is also possible for truthtable kernelizations. In particular, we exemplary consider Clique parameterized by the maximum c of the vertex-connectivities[34] of each induced subgraph of the input graph G. We show that c-Clique admits a non-trivial truthtable kernelization with $|V(G)|^2 / f(c)$ queries, each of size $f(c)$ for a function $f : \mathbb{N} \to \mathbb{N}$ with $\forall_x f(x) \geq 2x$. Using a balanced cut instead of just any cut allows us to reduce the number of queries at the cost of increased query sizes.

5.3.1 Trading Parallelism for Query Number

Assume we have developed a poly(k)-size poly(k)-evaluable truthtable kernelization but we are in the unfortunate situation that we do not have access to parallel

[34] The vertex-connectivity of a graph is the minimum number of vertices to delete to disconnect it.

computation facilities. Instead of serializing our truthtable kernelization, we might as well exploit the additional power that Turing kernelizations offer to save some of the queries. Thus, a framework for transforming truthtable kernelizations into Turing kernelizations might be handy in such situations.
In a second scenario, suppose we have designed a polynomial-size Turing kernelization posing only few queries, but we have a large array of CPUs at our disposal. A way of turning this Turing kernelization into a polynomial-size truthtable kernelization might be desirable, even if it poses exponentially more queries.
The following theorem shows that, under certain restrictions to the number of queries, both conversions are possible. It uses the concept of "mind changes" introduced by Beigel [15] for truthtable reductions.

Theorem 5.1. *Let k-Q be an* NP*-complete parameterized problem and let m be polynomial in k. Then, k-$Q \in \mathrm{TT}_m^{\mathrm{poly}(k)} \Leftrightarrow k\text{-}Q \in \mathrm{Tu}_{\log m}$.*

Proof (Largely following ideas of Beigel [15], Theorem 4.1). We show both directions separately, pointing out differences to [15] where appropriate.
"$\Leftarrow$": Let k-$Q \in \mathrm{Tu}_{\log m}$. A poly($k$)-size m-query truthtable kernel for k-Q can simply construct all possible sequences of instances that the Turing kernelization can construct. Since the Turing kernel cannot construct more than $\log m$ instances of k-Q with each computation, such a truthtable kernel constructs at most $2^{\log m} = m$ instances. Each computation of the Turing kernel corresponds to a vector of $\log m$ bits representing the answers to the $\log m$ queries $(q_1, q_2, \ldots)$. Let the decision of a computation corresponding to the vector $y \in \{0,1\}^{\log m}$ be $c_y \in \{0,1\}$. Then, we build a size-m CNF formula for the truthtable kernel as follows: For each $y \in \{0,1\}^{\log m}$ with $c_y = 0$, add a clause that, for each $1 \leq i \leq \log m$, contains q_i if $y_i = 0$ and $\neg q_i$ if $y_i = 1$. The constructed instance then consists of m clauses, each of size at most $\log m$. Since $m \in \mathrm{poly}(k)$, this implies k-$Q \in \mathrm{TT}_m^{\mathrm{poly}(k)}$.
"$\Rightarrow$": Let k-$Q \in \mathrm{TT}_m^{\mathrm{poly}(k)}$. Similar to the proof of Beigel [15] for reductions between NP-problems, we employ the "mind-change" technique to show this result. The idea is to "track the development" of the evaluation formula eval_x with growing number of answered queries, assuming an answer of "No" to unanswered queries. That is, in the beginning, all queries are assumed to not be in k-Q. Then, step by step, the correct answers to the queries "become known" to eval_x, causing it to change the value it evaluates to, it "changes its mind". Now, if we knew the number of mind changes and the value of $\mathrm{eval}_x(0,0,\ldots)$, then we would also know the value of $\mathrm{eval}_x(\chi^m_{k\text{-}Q}(\mathcal{I}_x))$. As it turns out, the number of mind changes can be computed with at most $\log m$ queries into a language $T \in \mathrm{NP}$. Therefore, a Turing kernel can compute this number in polynomial time and return the final

value of the evaluation function. Informally, T is the set of pairs (x,t) such that eval_x makes at least t mind changes on input x. Then, the maximum number t such that $(x,t) \in T$ can be computed by binary search, requiring at most $\log m$ queries.

In the following, let $(\mathcal{I}_x, \text{eval}_x)$ denote a poly(k)-size m-query truthtable kernel for k-Q such that $|\text{eval}_x| \in \text{poly}(k)$ for each input (x,k). Furthermore, let $I_i \in \mathcal{I}_x$, $0 \leq i < m$ denote the i'th instance computed by the truthtable kernelization on input (x,k). For vectors $v, w \in \{0,1\}^m$, we define $v \prec w \Leftrightarrow (\forall_i\ v_i \leq w_i \wedge \exists_j\ v_j < w_j)$. The set of possible mind changes of $(\mathcal{I}_x, \text{eval}_x)$ is

$$T = \Big\{(x,t) : \exists_{v^0,\ldots,v^t \in \{0,1\}^m} \forall_{0\leq i<m} (v_i^t = 1 \Rightarrow I_i \in k\text{-}Q) \wedge \tag{5.1}$$

$$\forall_{0<j\leq t} (v^{j-1} \prec v^j) \wedge \tag{5.2}$$

$$\forall_{0<j\leq t} (\text{eval}_x(v^{j-1}) \neq \text{eval}_x(v^j))\Big\}. \tag{5.3}$$

Herein, the value v_i^j corresponds to the knowledge about the i'th query in step j of the process of "gradually gaining knowledge" described at the beginning. To ensure that we gain knowledge in each step, (5.2) forces each v^j to contain more 1s then v^{j-1}. With (5.3), we ensure that the evaluation formula changes its mind in each step. Finally, (5.1) ensures that, in the last step, each query is correctly represented in v_i^j. Note that, for the following reason, it suffices to require the implication shown in (5.1): If for some i, $v_i^t = 0$ and $I_i \in k$-Q, then $v_i^j = 0$ for all j. Then, consider an additional step represented by v^{t+1} with $v_i^{t+1} = 1$. If this step causes a mind change in eval_x, then $(x, t+1) \in T$ and, thus, eval_x makes at least t mind changes. If this step does not cause a mind change, then eval_x makes t mind changes independent from the answer to the i'th query. We conclude that $(x,t) \in T$ if and only if eval_x makes at least t mind changes in the process of "gradually gaining knowledge" that we described in the beginning.

For Beigel [15], it was enough to show that $T \in$ NP, which can be seen by considering the certificate $v^0, \ldots, v^t$. However, here, we have to prove that $(x,t) \in T$ can be decided by $\log m$ queries of size poly(k) into the language k-Q. To this end, we reduce T to SAT and note that the images of the reduction have size polynomial in k. Given an instance (x,t) of T, we build a SAT-instance in four steps. The formula contains the $m \cdot (t+1)$ variables v_i^j with $0 \leq i < m$ and $0 \leq j \leq t$. It is the conjunction of the following terms.

First, we model (5.1). For each $0 \leq i < m$, create the term $(v_i^t \Rightarrow I_i \in k\text{-}Q)$, where $I_i \in k$-Q is replaced by the SAT-formula given by the (many-one) reduction

of k-Q to SAT (recall that k-$Q \in$ NP). Since the queries of the truthtable kernel have size poly(k), we conclude that $|I_i| \in$ poly(k). Since $(v_i^t \Rightarrow (I_i \in k\text{-}Q))$ can be modeled as $(\neg v_i^t \vee (I_i \in k\text{-}Q))$, all these terms have size poly(k). Since $m \in$ poly(k), the number of these terms is also in poly(k).

Second, we model (5.2). For each $0 < j \leq t$, create

$$\left(\bigwedge_{0 \leq i < m} (v_i^{j-1} \Rightarrow v_i^j)\right) \wedge \left(\bigvee_{0 \leq i < m} \neg(v_i^{j-1} \Leftarrow v_i^j)\right).$$

This term models $(v^{j-1} \prec v^j)$ for all $0 < j \leq t$. By analogous arguments as above, neither the size nor the number of these terms exceeds poly(k).

Third, we model (5.3). For each $0 < j \leq t$, create the term $\neg(\text{eval}_x(v^{j-1}) \Leftrightarrow \text{eval}_x(v^j))$. Since the evaluation formula eval_x has size poly(k) and there are at most $t + 1 \leq k + 1$ such terms, the number and size of these terms is in poly(k).

Finally, the complete SAT-instance consists of a conjunction of the created terms and, by the above arguments, the created SAT-instance has size poly(k).

Since k-Q is NP-hard, there is a polynomial-time many-one reduction of SAT to k-Q. Thus, there is a polynomial-time many-one reduction of T to k-Q and the images of the reduction have size poly(k).
For an input (x, k), a poly(k)-size Turing kernelization for k-Q computes $t_{\max} := \max_t\{(x, t) \in T\}$ using binary search on T posing poly(k)-size images of the many-one reduction of T to k-Q. Finally, the Turing kernelization returns the XOR of $\text{eval}_x(0, 0, \ldots, 0)$ and the least significant bit of $t_{\max}$. Since $t_{\max} \leq m$, at most $\log m$ queries are posed by the Turing kernelization. □

A special case of Theorem 5.1 is the following.

Corollary 5.1. *Let k-Q be an* NP*-complete parameterized problem. Then, k-$Q \in \text{TT}^{\text{poly}(k)}_{\text{poly}(k)} \Leftrightarrow k\text{-}Q \in \text{Tu}_{\text{O}(\log k)}$.*

Corollary 5.1 is useful for problems in $\text{TT}^{\text{poly}(k)}_{\text{poly}(k)}$ that do not admit polynomial-size classical kernelizations. Thus, an interesting open question posed implicitly by Theorem 5.1 is

> "Is there a parameterized problem that admits a polynomial-size truthtable kernel with polynomial-size evaluation formula but resists polynomial-size classical kernelization?"

Our search for a problem in $\mathrm{TT}^{\mathrm{poly}(k)}_{\mathrm{poly}(k)} \setminus \mathrm{DF}$ was rather unsuccessful so far. We can, however, show some properties of such a problem, provided one exists. More precisely, we show that, if the problem is NP-complete, then no poly(k)-size poly(k)-evaluable truthtable kernelization has a monotone—like the evaluation formulae of all truthtable kernelizations known to us—or antimonotone evaluation formula.

Proposition 5.1. *Let k-Q be an* NP*-complete parameterized problem that admits a* poly(k)*-size* poly(k)*-evaluable truthtable kernel $(\mathcal{I}_x, \mathrm{eval}_x)$ for k-Q such that $|\mathrm{eval}_x| \in \mathrm{poly}(k)$ and eval_x is monotone for each input (x, k). Then, k-$Q \in$ DF.*

Proof. We construct a many-one kernel from the truthtable kernel by replacing eval_x with a single query into k-Q. First, make sure that each variable in eval_x occurs exactly once by replacing a second occurance by a new variable with an equal query corresponding to it. Then, for each variable y_i, let "$(x_i, k_i) \in k$-Q?" denote the corresponding query.
Then, let ϕ_x be the quantified boolean formula resulting from replacing each occurrence of y_i in eval_x with $\exists_{z^i} \rho^i_x$, where ρ^i_x denotes a boolean formula on the variable-vector z^i such that ρ^i_x is satisfiable if and only if $(x_i, k_i) \in k$-Q. Since k-$Q \in$ NP, such a formula exists and can be computed in polynomial time. Let ρ_x denote the unquantified version (that is, the boolean formula that results from removing all quantifiers) of ϕ_x. Due to the monotonicity of eval_x, we conclude that

$$\phi_x \equiv \exists_{z^0, z^1, \ldots} \rho_x.$$

Since k-Q is NP-hard, there is a polynomial-time many-one reduction R of SAT to k-Q. Finally, the algorithm outputs $R(\rho_x)$.
It remains to show that the constructed algorithm is a polynomial-size classical kernelization for k-Q. First, since the size of each query is in poly(k), and the reduction we used to obtain ρ^i_x for all i is polynomial-time computable, we know that $|\rho^i_x| \in \mathrm{poly}(k)$. Then, by $|\mathrm{eval}_x| \in \mathrm{poly}(k)$, we conclude that $|\rho_x| \in \mathrm{poly}(k)$ and, therefore, $|R(\rho_x)| \in \mathrm{poly}(k)$.
Second, since ρ^i_x is satisfiable if and only if $\chi_{k\text{-}Q}((x_i, k_i)) = \mathrm{true}$, we know that ρ_x is satisfiable if and only if $\mathrm{eval}_x(\chi_{k\text{-}Q}(\mathcal{I}_x)) = \mathrm{true}$. Therefore, $R(\rho_x) \in k$-Q if and only if $\mathrm{eval}_x(\chi_{k\text{-}Q}(\mathcal{I}_x))) = \mathrm{true}$. □

Proposition 5.2. *Let k-Q be an* NP*-complete parameterized problem that admits a* poly(n)*-size* poly(n)*-query truthtable kernel $(\mathcal{I}_x, \mathrm{eval}_x)$ for k-Q such that eval_x is antimonotone for each input (x, k). Then,* NP = coNP.

Proof. Since each eval_x is antimonotone, there are monotone formulae ψ_x such that $\mathrm{eval}_x \equiv \neg\psi_x$ for all instances (x, k) of k-Q. Analogous to the proof of

Proposition 5.1, we can replace ψ by a single query into SAT and, hence, we can replace eval_x by a single query into TAUT, the set of all boolean formulae that are tautologies. Since the size of all queries and the number of queries is bounded polynomially in $|x|$, we have constructed a polynomial-time many-one reduction of k-Q to TAUT, implying k-$Q \in$ coNP and, since K-Q is NP-complete, NP = coNP follows. □

Proposition 5.1 and Proposition 5.2 impose restriction on the evaluation formulae of truthtable kernelizations for problems in $\text{TT}^{\text{poly}(k)}_{\text{poly}(k)} \setminus \text{DF}$. Moreover, to prove that a problem k-Q is in $\text{TT}^{\text{poly}(k)}_{\text{poly}(k)} \setminus \text{DF}$, we have to exclude polynomial-size classical kernelizations. As we will see in the next section, however, polynomial-size classical kernels for k-Q cannot be excluded using the established composition-based lower bound machinery, rendering the search for k-Q rather difficult.

5.3.2 Coarse Lower-Bound Tools

The question for lower bounds for truthtable or Turing kernelizations is, as this kind of preprocessing is rather new, largely open. Hermelin et al. [117] presented a reduction-based hardness-hierarchy approach that allows lower bounds for Turing kernelizations under the assumption that $(k \log n)$-Clique does not admit a polynomial-size Turing kernel. A promising candidate for a lower-bound framework that is based on a stronger conjecture may be the recently presented lower bounds machinery for oracle communication protocols by Dell and van Melkebeek [58]. In this section, we show that the available lower bound framework for classical kernelizations [26, 95, 58] is enough to exclude poly(k)-size poly(k)-evaluable truthtable kernelizations. Unfortunately, a downside of this is that these techniques are unfit to show that a problem in $\text{TT}^{\text{poly}(k)}_{\text{poly}(k)}$ does not admit a polynomial-size classical (many-one) kernelization.
In the following, we briefly state the concept of a co-nondeterministic kernelization (sometimes called "coNP kernel"). We will use it to show that certain problems do not admit poly(k)-size poly(k)-evaluable truthtable kernelizations. Intuitively, a co-nondeterministic kernel produces a number of small instances all of which must be yes-instances if the input was a yes-instance and vice versa.

Definition 5.4. *Let k-Q be a parameterized problem and let $g : \mathbb{N} \to \mathbb{N}$ be a function. Consider a nondeterministic polynomial-time algorithm A that, given an instance (x, k) of k-Q, produces an instance (x_i, k_i) on each nondeterministic computation path such that*

1. *$|x_i| \leq g(k)$ and $k_i \leq k$ for all i and*
2. *$(x, k) \in k\text{-}Q \Leftrightarrow \forall_i\ (x_i, k_i) \in k\text{-}Q$.*

Then, we call A a co-nondeterministic kernelization of size $g(k)$ for k-Q.

Co-nondeterminism for kernelization lower bounds was recently considered by Kratsch [136] who, slightly differing from our setup, allows the composition algorithm to be co-nondeterministic and considers deterministic kernelization. As observed before [193], the composition-based framework [26, 95, 58] also works if both the kernelization and the composition algorithm are co-nondeterministic.

Lemma 5.1. *Let k-Q be an* NP*-complete parameterized problem that is compositional. Furthermore, let k-Q admit a* poly(k)*-size co-nondeterministic kernelization. Then,* NP $\subseteq$ coNP/ poly.

The proof is exactly the same as the proof of Lemma 3.2 of Kratsch [136] with the difference that the co-nondeterminism is used in the kernelization instead of the composition. Therefore, we omit the proof here.
Lemma 5.1 directly implies that, assuming NP $\not\subseteq$ coNP/ poly, there is no conjunctive truthtable kernelization with monotone evaluation formula for compositional problems. In the following, we show that, under the above assumption, no NP-hard parameterized problem in $\mathrm{TT}^{\mathrm{poly}(k)}_{\mathrm{poly}(k)}$ is compositional.

Theorem 5.2. *Let k-Q be an* NP*-complete parameterized problem in* $\mathrm{TT}^{\mathrm{poly}(k)}_{\mathrm{poly}(k)}$*. Then, k-Q admits a* poly(k)*-size co-nondeterministic kernelization.*

Proof. Let $(\mathcal{I}_x, \mathrm{eval}_x)$ be a poly(k)-size truthtable kernelization for k-Q such that $|\mathrm{eval}_x| \in \mathrm{poly}(k)$ for all instances (x, k) of k-Q. Given (x, k) and $(\mathcal{I}_x, \mathrm{eval}_x)$, we construct a poly($k$)-size co-nondeterministic kernelization for k-Q in poly($|x| + k$) time.
First, assume that each variable of eval_x occurs exactly once in eval_x (if a variable occurs twice, we can just pose the same query twice, thus posing poly(k) additional queries). For each query $(x_i, k_i) \in \mathcal{I}_x$, let y_i denote the variable representing the answer to the question "$(x_i, k_i) \in k$-Q?". Furthermore, let ψ_i denote a poly(k_i)-size SAT-instance on the variable-vector z^i such that $\psi_i \in \mathrm{SAT} \Leftrightarrow (x_i, k_i) \in k$-$Q$. Since k-$Q \in$ NP, such an instance is computable in poly($|x_i|$) time. Let ϕ_x denote the quantified boolean formula that results from replacing each y_i by $\exists_{z^i} \psi_i$ in eval_x. Observe that the size of ϕ_x is in poly(k) since $|\mathrm{eval}_x| \in \mathrm{poly}(k)$ and $|\psi_i| \in \mathrm{poly}(k)$ (since ψ_i is the result of a polynomial-time many-one reduction applied to a poly(k)-size query in $\mathcal{I}_x$).
Let ρ_x denote the unquantified version of ϕ_x, that is, the result of removing all quantifiers from ϕ_x. Then, since the variable-sets of each two of the created SAT-instances are disjoint, there are disjoint sets $A, B \subseteq \mathbb{N}$ with $A = \{a_0, a_1, \ldots\}$ and $B = \{b_0, b_1, \ldots\}$ such that

$$\phi_x \equiv \underset{z^{a_0}, z^{a_1}, \ldots}{\forall} \quad \underset{z^{b_0}, z^{b_1}, \ldots}{\exists} \rho_x, \tag{5.4}$$

that is, the prenex normal form of ϕ_x has only one quantifier alternation.
Then, a nondeterministic algorithm can branch into a different computation path for each assignment of all z^a with $a \in A$. Let $\beta_x^p : \{z^a : a \in A\} \to \{\text{false}, \text{true}\}$ denote such an assignment on computation path p. On p, the formula ϕ_x degenerates to

$$\phi_x^p \equiv \underset{z^{b_0}, z^{b_1}, \ldots}{\exists} \rho_x^p, \tag{5.5}$$

where ρ_x^p is the boolean formula that results from ρ_x by replacing all occurrences of an element z_a^ℓ of z^a with $a \in A$ by the truth value assigned to it by β_x^p (that is, $\beta_x^p(z^a)^\ell$). Observe that $\rho_x^p \in \text{SAT} \Leftrightarrow \phi_x^p$ evaluates to true. Since k-Q is NP-complete, there is a polynomial-time reduction R of SAT to k-Q. Then, on each computation path p, the algorithm outputs $R(\rho_x^p)$.
In the following, we show that the algorithm we constructed is a co-nondeterministic kernelization for k-Q. Since $|\phi_x| \in \text{poly}(k)$, it follows that $|R(\rho_x^p)| \in \text{poly}(k)$ and, thus, Definition 5.4(1) holds. For Definition 5.4(2), note that, by correctness of the reduction of k-Q to SAT, $\text{eval}_x(\chi_{k\text{-}Q}(\mathcal{I}_x)) = \text{true}$ if and only if ϕ_x evaluates to true. Thus,

$$\begin{aligned}
\underset{p}{\forall} R(\rho_x^p) \in k\text{-}Q \quad &\Leftrightarrow \quad \left(\underset{p}{\forall} \phi_x^p\right) \\
&\Leftrightarrow \quad \left(\underset{z^{a_0}, z^{a_1}, \ldots}{\forall} \phi_x^p\right) \\
&\overset{(5.4),(5.5)}{\Leftrightarrow} \quad \phi_x \\
&\Leftrightarrow \quad \text{eval}_x(\chi_{k\text{-}Q}(\mathcal{I}_x)) = \text{true} \\
&\Leftrightarrow \quad (x,k) \in k\text{-}Q. \qquad \square
\end{aligned}$$

On the one hand, Lemma 5.1 and Theorem 5.2 let us use the existing lower bound framework for co-nondeterminisically compositional problems to show lower bounds for polynomial-size truthtable kernelizations with polynomial-size evaluation formulae. On the other hand, we cannot use compositions to exclude classical kernelizations for problems admitting such truthtable kernelizations. Therefore, to distinguish truthtable kernelizations with polynomial-size evaluation formulae from classical kernelizations, we need a more fine-grained tool than composition.

5.3.3 Trading Queries for Size: An Example using Clique

In this section, we consider another possible tradeoff for truthtable kernelizations. In particular, we show that some truthtable kernels allow a continuous tradeoff

Function CliqueTT(G, k)

Input: An instance (G, k) of Clique.
Output: A set of queries (G', k) into Clique such that each G' contains at most $f(c)$ vertices and G contains a k-clique if and only if some G' does.

```
1 if |V(G)| > f(c) then
2     X ← a minimum-cardinality vertex cut of G;
3     V_1 ← the vertex set of a smallest component of G − X;
4     V_2 ← V(G) \ V_1;
5     return CliqueTT(G[V_1 ⊎ X], k) ∪ CliqueTT(G[V_2], k);
6 else return {G};
```

between the number and the size of the queries. To this end, we consider Clique with respect to the parameter "maximum c of the vertex connectivities[35] of all subgraphs of G" (see Definition 5.5).
We first develop a truthtable kernelization posing at most n^2 queries, each containing at most $2c$ vertices and modify this kernelization to incorporate a tradeoff between the number of posed queries and their size. The modified truthtable kernelization poses $n^2/f(c)$ queries, each of size $f(c)$ for each computable function $f : \mathbb{N} \to \mathbb{N}$ with $f(c) \geq 2c$. In the rest of this section, we will implicitly assume that f is computable. Finally, we consider a balanced version of our parameter. This allows us to lower the quadratic bound on the number of queries to quasilinear at the cost of increasing the kernel size.
Formally, our parameter is defined as follows.

Definition 5.5. *A* cutset *(or* vertex cut*) X for G is a set of vertices such that $G - X$ is disconnected.*[36] *The smallest number c such that for each $V' \subseteq V$ there is a cutset $X' \subseteq V'$ with $|X'| \leq c$ for $G[V']$ is called the* subgraph-cutset-number.

Since each graph can be disconnected by deleting a smallest neighborhood, c is a stronger parameter than the "degeneracy[37] d of the input G". In turn, since the maximum degree of a subgraph of G cannot exceed the maximum degree of G, d is stronger than the maximum degree Δ of G (see Figure 1.1 on page 21; recall that Δ-Clique admits a linear-size truthtable kernelization [117]).

[35] The vertex connectivity (or vertex cut size) of a graph G is the smallest number of vertices whose deletion disconnects G.
[36] Following the literature [131], a graph with just one vertex is considered disconnected.
[37] The *degeneracy* (or *coloring number*) of a graph G is the smallest d such that all subgraphs of G contain a vertex of degree at most d [60].

The truthtable kernelization for c-Clique is presented as function CliqueTT. Intuitively, it finds a minimum-size cutset of a given graph and asks for a clique in the two subgraphs of the instance that are separated by this cutset. To be sure that a clique in the input is in at least one of the two subgraphs, we add the cutset to both of them. The implicit evaluation formula is simply a disjunction over all created instances. In line 2, a minimum vertex cut is computed. Even [80] describes how such a vertex cut can be computed for a graph with n vertices and m edges using n applications of a unit capacity maximum flow algorithm, each of which runs in $O(\sqrt{n}m)$ time. The following lemma is central to proving the correctness of function CliqueTT. It also shows that function CliqueTT can be used to enumerate all maximal cliques in the input.

Proposition 5.3. *Let K be a clique in a graph G and let $\mathcal{I}_G$ denote the instances computed by function CliqueTT on input (G,k). Then, there is some $(G',k') \in \mathcal{I}_G$ such that K is a subgraph of G'.*

Proof. We show the claim for one recursive step of function CliqueTT. The lemma then follows by induction. Let X denote the vertex cut then is computed in line 2 of function CliqueTT. If $V(G) \leq 2f(c)$, then function CliqueTT returns (G,k) and the claim of the lemma follows. Otherwise, let V_1 be the vertex set of a smallest component of $G' := G - X$ and let $V_2 := V(G) \setminus V_1$. Since X is a vertex cut, G does not contain an edge between any vertex in V_1 and any vertex in $V_2 \setminus X$. Hence, K is a subgraph of either $G[V_1 \uplus X]$ or $G[V_2]$. □

The following technical lemma is needed to prove the bound on the number of queries posed by function CliqueTT.

Lemma 5.2. *Let $c,n,\gamma \in \mathbb{N}^+$ with $\gamma \leq c$ and let $f : \mathbb{N} \to \mathbb{N}$ be a function with $f(c) \geq 2c$. For the recurrence $R(n) = R(n-x)+R(x+\gamma)$ subject to $\forall_{i \leq f(c)} R(i) = 1$ and $n - x \geq x + \gamma$, it holds that $R(n) \leq n^2/f(c)$.*

Proof. We prove the claim by induction over n. For $n = f(c)$, the statement clearly holds. Let $n > f(c)$ and assume that the statement holds for all $i < n$. We use the facts that

(i) $R(i) = 1$ for $i \leq f(c)$,
(ii) $R(i) \leq i^2/f(c)$ for $i < n$, and
(iii) $n - x \geq x + \gamma$

to prove each of the following cases.

Case 1: $n - x \leq f(c)$ and $x + \gamma \leq f(c)$. Then, $R(n) \overset{(i)}{=} 2 \overset{n > f(c)}{\leq} n^2/f(c)$.
Case 2: $n - x \leq f(c) < x + \gamma$. This contradicts (iii).

Case 3: $n - x > f(c) \geq x + \gamma$. Then,

$$R(n) \overset{(i)}{=} R(n-x) + 1 \overset{(ii)}{\leq} \frac{(n-x)^2}{f(c)} + 1 \leq \frac{n^2 - 2x(n-x-f(c))}{f(c)} \overset{n-x>f(c)}{\leq} \frac{n^2}{f(c)}.$$

Case 4: $n - x > f(c)$ and $x + \gamma > f(c)$. Then, $x > f(c) - \gamma \geq \gamma$ and

$$R(n) \overset{(ii)}{\leq} \frac{(n-x)^2}{f(c)} + \frac{(x+\gamma)^2}{f(c)} \overset{(iii)}{\leq} \frac{n^2}{f(c)} + \frac{-2x^2 + \gamma^2}{f(c)} \overset{x \geq \gamma}{\leq} \frac{n^2}{f(c)}. \qquad \square$$

Theorem 5.3. *Let $f : \mathbb{N} \to \mathbb{N}$ be a function such that $\forall_x f(x) \geq 2x$. Then,* Clique *parameterized by the subgraph-cutset-number c of the input graph admits an $n^2/f(c)$-query truthtable kernel containing at most $f(c)$ vertices. The kernel can be computed with $O(n^2/f(c))$ minimum vertex cut computations.*

Proof. We show that, given an instance (G, k) of Clique, function CliqueTT computes a set of instances $\mathcal{I}_G$ that, together with the evaluation formula that is a disjunction over its $|\mathcal{I}_G|$ variables, this constitutes the claimed truthtable kernelization.

First, we show that function CliqueTT constructs at most $n^2/f(c)$ queries, each containing at most $f(c)$ vertices. If $n \leq f(c)$, then G is already a kernel and the claimed size holds. Otherwise, $n > f(c)$. Let X, V_1, and V_2 be the sets computed by function CliqueTT in lines 2, 3, and 4. Since $V(G) = V_1 \uplus V_2 \uplus X$, line 3 implies that $|V_1| \leq (n - |X|)/2$. Since X is a cutset for G, we conclude $|X| \leq c$. Then, the number of created instances is

$$R(n) = R(n - |V_1|) + R(|V_1| + |X|) \text{ subject to } \forall_{i \leq f(c)} R(i) = 1.$$

By choice of V_1 in line 3, $|V_1| \leq |V_2|$. Since $|V_1| + |V_2| + |X| = n$, we further conclude that $|V_1| + |X| \leq n - |V_1|$. Now, Lemma 5.2 implies $R(n) \leq n^2/f(c)$. Furthermore, since line 6 is the only line returning instances, the number of vertices in each instance is bounded by $f(c)$.

It remains to show that $(\mathcal{I}_G, \text{eval}_G)$ is indeed a truthtable kernel, that is, G contains a k-clique if and only if $\mathcal{I}_G$ contains a pair $(G', k) \in$ Clique. The "$\Leftarrow$"-direction is implied by the fact that all graphs of instances in $\mathcal{I}_G$ are induced subgraphs of G. The "$\Rightarrow$"-direction is a direct consequence of Proposition 5.3. $\square$

Note that Theorem 5.3 implies a parameterized algorithm solving c-Clique in $O((2^{f(c)} \cdot n + T_C(n,m)) \cdot n^2/f(c))$ time for any function $f : \mathbb{N} \to \mathbb{N}$ with $f(c) \geq 2c$ and $T_C(n,m) \in O(n^{1.5} \cdot m)$ denoting the time needed to compute a minimum vertex cut in a graph with n vertices and m edges [80].

By Proposition 5.3, function CliqueTT can be used to enumerate all maximal cliques in G. Thus, our truthtable kernelization with respect to c extends previous work by Eppstein et al. [78], who showed an algorithm that enumerates all maximal cliques in $O(3^{d/3} \cdot dn)$ time, where d denotes the degeneracy of the input graph.

Corollary 5.2. *Let G be a graph with subset-cutset number c. Then, all maximal cliques of G can be enumerated in $O((4^c \cdot c^2 + \sqrt{nm}) \cdot n^2/c)$ time.*

In ongoing work [151], we improved the presented truthtable kernelization to pose only $n - f(c)$ queries, each containing at most $f(c) + 1$ vertices. Then, the running time of the implied enumeration algorithm for Clique becomes $O((2^c \cdot c^2 + \sqrt{nm}) \cdot (n - c))$. Furthermore, it turns out that $c \leq d \leq 2c$ for all graphs, which is tight: On the one hand, Eppstein et al. [78] constructed graphs whose degeneracy d equals our parameter c; on the other hand, there are arbitrarily large connected graphs with $d = 2c$ [151]. Since $3^{d/3} > 2^{0.5d}$, algorithms based on the truthtable kernelization presented in this section might outperform the one presented by Eppstein et al. [78] on some graphs. Thus, a comparison of the algorithms in an application setting may be an interesting candidate for future work.

Reducing the number of queries using balance. In the following, we show that the number of posed queries can be reduced to quasilinear in n. However, this comes at the cost of having to pick a slightly weaker parameter than the subset-cutset size c. Using a regular cutset to split the input graph may yield a highly unbalanced distribution of vertices. Therefore, it is hard to circumvent the quadratic bound on the number of queries in our strategy. Hence, we consider a vertex cut X separating V_1 from V_2 such that $|V_1|$ is within a factor α of $|V_2|$.

Definition 5.6. *Let X be a cutset of a graph G and let $\alpha \in \mathbb{N}^+$. If there are vertex sets V_1 and V_2 such that*

(i) $V_1 \uplus V_2 \uplus X = V(G)$,
(ii) *there are no edges between V_1 and V_2 in G, and*
(iii) $|V_1| \leq |V_2| \leq \alpha|V_1|$,

*then we call X α-*balanced*. The maximum over the minimum sizes of α-balanced cutsets of all subgraphs of G is called the α-*balanced subgraph cutset number $\hat{c}_\alpha$. A δ-*vertex separator *of G is a vertex set X such that each connected component of $G - X$ contains at most $\delta \cdot n$ vertices.*

Note that, for all α, an α-balanced vertex cut is always larger than the minimum vertex cut. Thus,

$$\forall_\alpha \; c \leq \hat{c}_\alpha \leq \hat{c}_{\alpha+1}.$$

As it turns out, finding an α-balanced vertex cut is NP-hard [149]. Hence, to compute an α-balanced subgraph cutset in polynomial time, we employ an approximation of a δ-vertex separator by Feige et al. [81]. The following lemma shows that the two notions of balanced separators are equivalent.

Lemma 5.3. *Let $X \subseteq V(G)$. Then, X is a δ-vertex separator of G for some $\delta < 1$ if and only if X is an α-balanced cutset of G for some α.*

Proof. We show both directions separately.
"$\Rightarrow$": Let X be a δ-vertex separator of G. Then we can partition the connected components of $G - X$ in two cells C_1 and C_2 with $n_1 := |V(C_1)|$ and $n_2 := |V(C_2)|$ such that $n_1 \leq n_2 \leq \delta \cdot n + n_1$. Since $n_1 + n_2 = n$, we derive $n_1 \geq n \cdot (1 - \delta)/2$, implying that X is an α-balanced cutset of G for

$$\alpha := \frac{n_2}{n_1} \leq \frac{\delta \cdot n + n_1}{n_1} = \frac{\delta \cdot n}{n_1} + 1 \leq \frac{\delta \cdot n}{\frac{1-\delta}{2} \cdot n} + 1 = \frac{1+\delta}{1-\delta}$$

"$\Leftarrow$": Let X be an α-balanced subgraph cutset of G. Then, $V(G) \setminus X = V_1 \uplus V_2$ such that there are no edges between vertices in V_1 and vertices in V_2 in $G - X$ and, with $n_1 := n_1$ and $n_2 := n_2$, we have $n_1 \leq n_2 \leq \alpha \cdot n_1$. Since $n_1 + n_2 = n$, we derive $n_2 \leq n \cdot \alpha/(\alpha + 1)$, implying that X is also a δ-vertex separator with

$$\delta := \frac{n_2}{n} \leq \frac{\alpha}{\alpha + 1}.$$ □

With Lemma 5.3, we can use the algorithm of Feige et al. [81] to compute an α-balanced cutset for G whose size is within a factor of $\log n$ of the optimal α-balanced cutset for G. By Theorem 5.3, Clique is solvable in polynomial time if the subgraph-cutset number c of G is smaller than $\log n$. Thus, we can find an α-balanced cutset of size $\hat{c}_\alpha \cdot \log n \leq \hat{c}_\alpha \cdot c \leq \hat{c}_\alpha^2$ in polynomial time. Then, replacing c by $\hat{c}_\alpha^2$ and the cutset X by an α-balanced cutset X' in function CliqueTT yields a truthtable kernel whose queries have size $f(\hat{c}_\alpha^2)$. To show that the number of queries is quasilinear in n, we prove a stricter version of Lemma 5.2 using Definition 5.6(iii).

Lemma 5.4. *Let $c, n, x, \gamma, \alpha \in \mathbb{N}^+$ such that (a) $\gamma \leq c$, (b) $n - x \leq \alpha x + \gamma$, and (c) $n \geq 2^{2\alpha+1}$. Let $f_\alpha : \mathbb{N} \to \mathbb{N}$ be a strictly increasing function with*

$$\forall_z f_\alpha(z) \geq 2z \quad \text{and} \quad \forall_{z > f_\alpha(c)} f_\alpha\left(\frac{z \cdot \log \frac{\alpha+2}{\alpha+1}}{\log z}\right) \geq z. \tag{5.6}$$

For the recurrence $R(n) = R(n-x) + R(x+\gamma)$ subject to $\forall_{i \leq f_\alpha(c)} R(i) = 1$ and $n - x \geq x + \gamma$, it holds that $R(n) \leq n \log n / f_\alpha(c)$.

Proof. First, let f^{-1} denote the inverse of f and note that, since f is strictly increasing, so is f^{-1}. We prove the claim by induction over n. For $n = f_\alpha(c)$, the statement clearly holds. Let $n > f_\alpha(c)$ and assume that the statement holds for all $i < n$. We use the facts that

(i) $R(i) = 1$ for $i \leq f_\alpha(c)$,
(ii) $R(i) \leq i \log i / f_\alpha(c)$ for $i < n$, and
(iii) $n - x \geq x + \gamma$

to prove each of the following cases.

Case 1: $n - x \leq f_\alpha(c)$ and $x + c \leq f_\alpha(c)$. Then, $R(n) \overset{(i)}{=} 2 \overset{n > f_\alpha(c)}{\leq} n \log n / f_\alpha(c)$.
Case 2: $n - x \leq f_\alpha(c) < x + \gamma$. This contradicts (iii).
Case 3: $n - x > f_\alpha(c) \geq x + \gamma$. Then, $n - x > 2c \geq 2\gamma$. Furthermore,

$$\begin{aligned} 2(\alpha+1)(n - x(\log n + 1)) &\overset{(b)}{\leq} 2(\alpha+1)n - 2(n-\gamma)(\log n + 1) \\ &= -2n(\log n - \alpha) + 2\gamma(\log n + 1) \\ &\overset{n \geq 2\gamma}{\leq} -n(\log n - (2\alpha + 1)) \overset{(c)}{\leq} 0. \end{aligned} \tag{5.7}$$

Then, we conclude

$$\begin{aligned} R(n) &\overset{(i)}{=} R(n-x) + 1 \overset{(ii)}{\leq} \frac{(n-x)\log(n-x)}{f_\alpha(c)} + 1 \\ &\overset{n-x > f_\alpha(c)}{\leq} \frac{(n-x)(\log n + 1)}{f_\alpha(c)} \leq \frac{n \log n}{f_\alpha(c)} + \frac{n - x(\log n + 1)}{f_\alpha(c)} \overset{(5.7)}{\leq} \frac{n \log n}{f_\alpha(c)} \end{aligned}$$

Case 4: $n - x > f_\alpha(c)$ and $x + \gamma > f_\alpha(c)$. Then, $x \geq f_\alpha(c) - \gamma + 1 \geq \gamma + 1$. Furthermore, since f_α^{-1} is strictly increasing,

$$\frac{n \log \frac{\alpha+2}{\alpha+1}}{\log n} \overset{(5.6)}{\geq} f_\alpha^{-1}(n) \overset{n > f_\alpha(c)}{>} f_\alpha^{-1}(f_\alpha(c)) = c \tag{5.8}$$

Then, we conclude

$$\begin{aligned} R(n) &\overset{(ii)}{\leq} \frac{(n-x)\log(n-x)}{f_\alpha(c)} + \frac{(x+\gamma)\log(x+\gamma)}{f_\alpha(c)} \\ &= \frac{n \log n}{f_\alpha(c)} + \frac{-n \log \frac{n}{n-x} + x \log \frac{x+\gamma}{n-x} + \gamma \log(x+\gamma)}{f_\alpha(c)} \\ &\overset{(iii)}{\leq} \frac{n \log n}{f_\alpha(c)} + \frac{-n \log \frac{n}{n-x} + \gamma \log(x+\gamma)}{f_\alpha(c)} \\ &\overset{(a),(iii)}{\leq} \frac{n \log n}{f_\alpha(c)} + \frac{-n \log \frac{\alpha+2}{\alpha+1} + c \log n}{f_\alpha(c)} \overset{(5.8)}{\leq} \frac{n \log n}{f_\alpha(c)} \end{aligned}$$

□

Note that, for all $\alpha \in \mathbb{N}^+$, there is a function $f_\alpha(z) \in \mathrm{O}(z^2)$ respecting (5.6) since $(z/\log z)^2$ grows asymptotically faster than z. Furthermore, since α is constant, instances with $n < 2^{2\alpha+1}$ have constant size and can be solved in polynomial time.
Plugging Lemma 5.4 into Theorem 5.3, then yields the desired decrease in the number of queries from quadratic in n to quasilinear in n. As a downside, however, we have to accept an increase in kernel size due to the hardness of the computation of balanced separators and the fact that balanced separators are larger than vertex cuts.

Corollary 5.3. *Let $\alpha \in \mathbb{N}^+$ and let $f : \mathbb{N} \to \mathbb{N}$ be a function such that (5.6) holds. Then,* Clique *parameterized by the α-balanced subgraph-cutset number $\hat{c}_\alpha$ of the input graph admits a truthtable kernelization that poses at most $(n \log n)/f(\hat{c}_\alpha^2)$ queries, each containing at most $f(\hat{c}_\alpha^2)$ vertices.*

5.4 Conclusion

We considered truthtable kernelization, a new type of kernelization based on Turing kernelization. On the one hand, truthtable kernelizations are "stronger", that is, harder to obtain than Turing kernelizations, but exhibit excellent parallelizability, which is an essential advantage as today's computational infrastructure provides massive parallelism capabilities. Furthermore, all published Turing kernelizations that we are aware of are, in fact, truthtable kernelizations, indicating that Turing kernelization might be an unnecessarily permissive tool for the considered problems. On the other hand, truthtable kernelization is "weaker" than classical (many-one) kernelization, that is, it allows preprocessing with performance guarantee for problems that do not admit polynomial-size classical kernelizations [88, 117, 172].
We showed that certain polynomial-size Turing kernelizations can be constructed from polynomial-size truthtable kernelizations with polynomial-size evaluation formula and vice versa. This result can be used to "trade in" parallel execution capabilities to decrease the number of queries posed by the kernelization.
We consider it an interesting open question whether $\mathrm{TT}^{\mathrm{poly}(k)}_{\mathrm{poly}(k)} = \mathrm{DF}$. If this is true, then we can use the power of truthtable kernelizations to show the existence of polynomial-size classical kernelizations. If $\mathrm{TT}^{\mathrm{poly}(k)}_{\mathrm{poly}(k)} \neq \mathrm{DF}$, then (co-nondeterministic) composition algorithms exclude more than just polynomial-size classical kernelizations. In this case, it would be desirable to have a more fine-grained tool to show lower bounds for preprocessing since polynomial-size polynomial-query truthtable kernels might be a practical approach to solving some parameterized problems that do not admit classical kernelizations. Either

way, an answer to the question whether $\mathrm{TT}^{\mathrm{poly}(k)}_{\mathrm{poly}(k)} = \mathrm{DF}$ entails a good amount of exciting research.

Many other open questions remain in the context of truthtable and Turing kernelizations. For example, is there a problem that requires the power provided by Turing kernelizations, that is, is there a problem admitting a polynomial-size Turing kernel but no polynomial-size truthtable kernel? What parameterized problems that are known to not admit polynomial-size classical kernels (under the hypothesis that $\mathrm{NP} \not\subseteq \mathrm{coNP/poly}$) have polynomial-size truthtable kernels? Can we circumvent the polynomial lower bounds established by the framework of Dell and van Melkebeek [58] or Hermelin and Wu [116] using truthtable or Turing kernelization? For example, is there an $\mathrm{O}(k)$-*size* truthtable kernel for k-Vertex Cover? Even if no polynomial lower bound is known, can we improve the sizes of known classical kernelizations by allowing more than one instance to be computed? In the spirit of efficient preprocessing, is it possible to "trade" queries or query sizes for faster kernelizations? An n-query truthtable kernel that can be computed in $\mathrm{O}(n^2)$ time might be more useful than a classical kernel that takes $\Omega(n^3)$ time to compute. Another interesting "trade" may be possible between queries and query size. Here, a linear-size n-query truthtable kernel might be theoretically and practically more relevant than a classical kernel of size $\mathrm{O}(k^3)$.

Chapter 6

Conclusion and Future Work

In this thesis, we analyzed NP-hard combinatorial (graph-) problems in the context of parameterized complexity. In particular, we shed light on important aspects of polynomial-time preprocessing that, to the best of our knowledge, have been largely neglected in the development and theoretical analysis of fixed-parameter algorithms.

6.1 The Thesis in a Nutshell

The aspects of preprocessing that we discussed were: structural (non-standard) parameters, efficient (linear-time) data reduction, preprocessing beyond kernelization, and truthtable kernelizations. Notably, these aspects can be combined freely, multiplying their respective effects. In the following, we will give a quick recapitulation of the respective chapters.

Non-standard parameters. The choice of parameter(s) is one of the most important choices when developing parameterized algorithms for combinatorial problems. There are several techniques for obtaining interesting and promising parameters ("deconstructing intractability", "distance from triviality", "above guarantee parameterization", ...) [160, 161], often yielding interesting parameters that are stronger than the "standard parameter" (solution size). Therefore, fixed-parameter algorithms for these parameters may outperform algorithms designed for the standard parameter.
The above holds true for solving a problem as well as preprocessing it. However, there are little kernelization results for structural parameters to date [30, 114, 123, 124, 125, 159]. We contributed to this list by developing a linear-size problem kernel for the Two-Layer Planarization problem parameterized by the feedback edge set number f of the input. As our parameter is provably smaller than the solution size, the algorithm constitutes an improvement over previous preprocessing algorithms. We complemented the problem kernel with an algorithm running in $O(3.8^f \cdot f^2 + f(n+m))$ time.

There is a seemingly endless supply of problems for which non-standard parameter make sense but have yet to be considered, especially regarding efficient preprocessing. Another opportunity for future work is the charting of the parameter landscape to discover more correllations between parameters. This is not limited to graph problems but is also interesting for problems on strings or numbers, which offer a wide array of parameters as well.

Efficiency in Preprocessing. Most results in parameterized algorithmics concern the running time of a parameterized algorithm solving the problem at hand or the size of a problem kernel that can be obtained in polynomial time. Some practical applications, however, are so time-critical that anything slower than (almost) linear time is not acceptable. These applications might settle for an approximate solution, which could greatly benefit from a linear-time kernelization.

Fast preprocessing is also attractive for the design of exact algorithms, since kernelizations can be run consecutively. As an example, we gave a linear-time algorithm that computes a problem kernel of size $O(\gamma)$ for Dominating Set on planar graphs, where γ denotes the domination number of the input. Since the instance can be expected to be smaller after the preprocessing, further kernelization algorithms [46] can be run on that instance. In this way, the best known (measured in the size of the produced kernel) kernelization algorithms are sped up by prepending a linear-time kernelization.

In this context, "fast" is not necessarily equivalent to linear or quasilinear time. An algorithm running in $O(k|G|)$ time can, under the hypothesis that the parameter is small in the input instance, also be considered efficient. In this sense, the kernelization developed in Section 2.3 is fast, since it is computable in $O(f|G|)$ time, where f is the feedback edge set number of the input.

Depending on the available hardware, "fast" can also mean that an algorithm scales well, that is, allows massive parallel execution. The possibility of easy work distribution and load balancing is one of the reasons why branching algorithms are so popular [150]. Seeing that today's computational infrastructure emphasizes parallel execution more than ever ("cloud computing", "crowd computing", GPU and multicore CPU programming), it is important to design easily parallelizable algorithms. As there are little results concerning parallel execution of kernelization algorithms, this opens an interesting field of study.

It is worth mentioning that fast preprocessing algorithms are not only interesting for NP-hard problems but also polynomial-time solvable problems, as our introductory example (see Section 1.1) involving a sorted array that can be queried for containment of some item in $O(\log n)$ time demonstrates.

Preprocessing Beyond Kernelization. Not admitting a polynomial-size problem kernel is often described as "incompressibility" or "limit of preprocessing". Is is, however, important to keep in mind that preprocessing is more than just kernelization and, for practical implementations, exponential-size problem kernels, preprocessing designed for another parameter, or even heuristic data reduction rules are important to simplify an input instance. In this sense, every bit of preprocessing counts.
We showed that a preprocessing algorithm that just alters the weights of arcs in an instance of k-Weighted Multigraph Eulerian Extension is capable of speeding up a dynamic programming algorithm that solves the problem. The running time was improved from $O(n^k 2^k \cdot n^4)$ to $O(4^k \cdot n^3)$. The preprocessing also allowed us state an upper bound on the parameter k in in terms of the parameters "number of connected components" and "number of imbalanced vertices" (see Theorem 4.4 on page 135).
In this context, we may reiterate our argument above, stating that modern computation is largely about parallel execution. For example, heuristic preprocessing algorithms that "split" the input instance into smaller parts can be considered a preprocessing. Although this "reduction" is unlikely to yield a kernelization by itself, it often allows parallel processing of the created components.

Between Turing and classical kernelization. Using composition algorithms, many parameterized problems have been shown to not admit polynomial-size kernelizations (as we have done for $(c + s)$-Switch Set Cover in Section 4.6, implying compositionality for k-Weighted Multigraph Eulerian Extension). For these problems, a more general definition of kernelization that captures the strategy of creating multiple small instances instead of just one, may be helpful. The concept of "Turing kernelization" is very young and there are only a handful published Turing kernelizations. As it turns out, however, none of these results make use of the full power of Turing kernelization.
We derived the concept of "truthtable kernelization" from the concept of truthtable reductions in recursion theory. A truthtable kernel is tailored to support massive parallel execution which, as discussed above, is especially important in the modern age of parallel computation. We showed the some properties of truthtable kernels can be traded to better fit the application scenario.

6.2 What the Future Holds

Operating in a theoretic science, we have to frequently question the significance of our work. In this sense, it is important to keep in touch with the real world and

ponder the impact of our work on it—what can we, as Mike Fellows put it, "sell to the engineers"? This practical orientation often inspires great out-of-the-box thinking. The creation of parameterized complexity as a way around NP-hardness and the development of Turing kernelization as a way around incompressibility are just two examples of this. Thus it is important not to loose sight of the practical applications our results will be applied to. This is especially true for preprocessing, since kernelization and preprocessing theory is arguably the export hit of parameterized complexity for practical computing.

The list of aspects of preprocessing discussed in this thesis is, by no means, exhaustive. Even restricted to kernelization algorithms, there are numerous further concepts to explore:

1. Data reduction rules for graph problems on instances (G, k) with $k \in \mathbb{N}$ can make use of the value of k. The well-known "Buss kernel" [35] for k-Vertex Cover is probably the most prominent example. Now, if we wanted to compute the vertex cover number τ of an input graph G, the canonical way is to run an algorithm that answers "$(G, k) \in$ Vertex Cover?" for increasing values of k. Each time, the kernelization may produce a different result and thus, we have to rerun it. In contrast, if our kernelization would be independent of k (except maybe for a trivial "If $|G| \notin \mathrm{O}(k)$, then return a trivial no-instance.") then we would not have to rerun it. Such a kernelization is said to be "parameter independent". This aspect of kernelization seems to be important for practical implementations, justifying further examination.
2. Applying a kernelization procedure is usually not considered hurtful or destructive. While this is true for combinations with exact algorithms, preprocessing may have a negative impact on approximation. First, A kernelization with respect to a structural parameter may increase the value of an optimal solution arbitrarily, as long as the output instance is small (measured in the parameter). Hence, running an approximation algorithm on the output of a kernelization may yield a much worse solution than running it on the input instance. Second, if the parameter is the sought solution size, then a kernelization might map a no-instance to another no-instance whose solution value is arbitrarily large, even though the new parameter value is bounded in the old parameter value [144]. Although kernelizations for many problems parameterized by the sought solution size can be turned into approximation preserving kernelizations [144], further considerations in this direction may be warrented.
3. In Chapter 5, we briefly mentioned the concept of α-fidelity kernelization [85]. This concept captures the idea that if we only want to compute an approximation, then we may allow making suboptimal decisions in the

kernelization, as long as their consequences are within the limits of the approximation ratio we are aiming at. This concept of "lossy" kernelization may also be interesting in the context of running time considerations: A linear-time 2-fidelity kernelization may be more interesting than a cubic time classical kernelization.

4. We call a set of reduction rules *confluent* if the order of their application does not influence the final result. This notion was studied by Ehrig et al. [74]. They "believe that to analyze whether a set of data reduction rules is confluent is a well-motivated and natural theoretical question of practical relevance with the potential for numerous opportunities for (interdisciplinary) future research between so far unrelated research communities". In this sense, considering the confluence of data reduction rules seems an interesting future research topic.

In the future, we hope to see multivariate (Turing/truthtable) kernelizations used up to their full potential. Problems of central importance that escaped classical kernelization so far are canonical candidates for this approach. What about, for example, Dominating Set? Parameterized by the vertex-deletion distance to clique, Dominating Set straightforwardly reduces to Red/Blue Dominating Set parameterized by the number of blue vertices, which is fixed-parameter tractable [61], but does it admit a polynomial-size (truthtable) kernel? What about the parameter "vertex cover number τ" or a combination of the two parameters? There are correct data reduction rules for Dominating Set (in fact, the rules presented in Chapter 3 are correct for general graphs) that just do not yield a kernel for the standard parameter. In this sense, instead of thinking of Dominating Set as a very hard problem, we advertise thinking of the domination number as a very strong (small) parameter.

Backed up by the recent mushrooming of kernelization results (upper *and* lower bounds), we conjecture that we have only seen the tip of the preprocessing iceberg. Practically relevant research is unlikely to get around the aspects of preprocessing that we discussed in this thesis. Therefore, our work hopefully helps designing algorithms that sell!

Bibliography

[1] *ILOG CPLEX user's manual*, 2009. URL ftp://public.dhe.ibm.com/software/websphere/ilog/docs/optimization/cplex/ps_usrmancplex.pdf. (Cited on page 2.)

[2] Website of the Clay Mathematics Institute, Accessed July 2012. URL http://www.claymath.org/millennium/. (Cited on page 11.)

[3] Table of FPT races, Accessed July 2012. URL http://fpt.wikidot.com/fpt-races. (Cited on page 5.)

[4] Scott Aaronson. The complexity zoo, Accessed July 2012. URL http://qwiki.stanford.edu/wiki/Complexity_Zoo/. (Cited on page 10.)

[5] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993. (Cited on pages 120, 121 and 123.)

[6] Ravindra K. Ahuja, Thomas L. Magnanti, James B. Orlin, and Charu C. Aggarwal. *Solution Manual for "Network Flows: Theory, Algorithms, and Applications"*, 2012. URL http://jorlin.scripts.mit.edu/Solution_Manual.html. (Cited on pages 120 and 121.)

[7] Jochen Alber, Hans L. Bodlaender, Henning Fernau, Ton Kloks, and Rolf Niedermeier. Fixed parameter algorithms for Dominating Set and related problems on planar graphs. *Algorithmica*, 33(4):461–493, 2002. (Cited on page 71.)

[8] Jochen Alber, Michael R. Fellows, and Rolf Niedermeier. Polynomial-time data reduction for Dominating Set. *Journal of the ACM*, 51(3):363–384, 2004. (Cited on pages ii, v, viii, 5, 8, 69, 70, 71, 72, 74, 75, 76, 77, 78, 79, 80, 88, 103, 104 and 105.)

[9] Jochen Alber, Nadja Betzler, and Rolf Niedermeier. Experiments on data reduction for optimal domination in networks. *Annals of Operations Research*, 146(1):105–117, 2006. (Cited on page 71.)

[10] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, NY, USA, 2009. (Cited on pages 9, 10, 11, 143 and 182.)

[11] Giorgio Ausiello, Pierluigi Crescenzi, Giorgio Gambosi, Viggo Kann, Alberto Marchetti-Spaccamela, and Marco Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer, 1999. (Cited on page 12.)

[12] Vineet Bafna, Piotr Berman, and Toshihiro Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM Journal on Discrete Mathematics*, 3(2): 289–297, 1999. (Cited on page 70.)

[13] Reuven Bar-Yehuda, Dan Geiger, Joseph Naor, and Ron M. Roth. Approximation algorithms for the feedback vertex set problem with applications to constraint satisfaction and Bayesian inference. *SIAM Journal on Computing*, 27(4):942–959, 1998. (Cited on page 70.)

[14] Ann Becker, Reuven Bar-Yehuda, and Dan Geiger. Randomized algorithms for the Loop Cutset problem. *Journal of Artificial Intelligence Research*, 12:219–234, 2000. (Cited on page 108.)

[15] Richard Beigel. Bounded queries to SAT and the Boolean hierarchy. *Theoretical Computer Science*, 84:199–223, 1991. (Cited on pages ix, 146 and 147.)

[16] Stéphane Bessy, Fedor V. Fomin, Serge Gaspers, Christophe Paul, Anthony Perez, Saket Saurabh, and Stéphan Thomassé. Kernels for feedback arc set in tournaments. *Journal of Computer and System Sciences*, 77(6):1071–1078, 2011. (Cited on pages 69 and 109.)

[17] Nadja Betzler, Jiong Guo, Christian Komusiewicz, and Rolf Niedermeier. Average parameterization and partial kernelization for computing medians. *Journal of Computer and System Sciences*, 77(4):774–789, 2011. (Cited on pages 6 and 140.)

[18] René van Bevern. Towards optimal and expressive kernelization for d-hitting set. In *Proceedings of the 18th Annual International Computing and Combinatorics Conference (COCOON'12)*, volume 7434 of *Lecture Notes in Computer Science*, pages 121–132. Springer, 2012. (Cited on pages 69, 70 and 106.)

[19] René van Bevern, Sepp Hartung, Frank Kammer, Rolf Niedermeier, and Mathias Weller. Linear-time computation of a linear problem kernel for dominating set on planar graphs. In *Proceedings of the 6th International Symposium on Parameterized and Exact Computation (IPEC '11)*, volume 7112 of *Lecture Notes in Computer Science*, pages 194–206. Springer, 2011. (Cited on pages viii and 70.)

[20] René van Bevern, Rolf Niedermeier, Manuel Sorge, and Mathias Weller. Complexity of arc routing problems. Submitted as a chapter in the book *Arc Routing: Problems, Methods and Applications* by Ángel Corberán and Gilbert Laporte (editors) that is to be published by SIAM. (Cited on page ix.)

[21] René van Bevern, Rolf Niedermeier, Matthias Mnich, and Mathias Weller. Interval scheduling and colorful independent sets. In *Proceedings of the 23rd International Symposium on Algorithms and Computation (ISAAC '12)*, volume 7676 of *Lecture Notes in Computer Science*, pages 247–256. Springer, 2012. (Cited on pages 5 and 20.)

[22] Robert E. Bixby. Solving real-world linear programs: A decade and more of progress. *Operations Research*, 50(1):3–15, 2002. (Cited on page 2.)

[23] Hans L. Bodlaender. Kernelization: New upper and lower bound techniques. In *Proceedings of the 4th International Workshop on Parameterized and Exact Computation (IWPEC '09)*, volume 5917 of *Lecture Notes in Computer Science*, pages 17–37. Springer, 2009. (Cited on pages 1 and 16.)

[24] Hans L. Bodlaender and Thomas C. van Dijk. A cubic kernel for feedback vertex set and loop cutset. *Theory of Computing Systems*, 46(3):566–597, 2010. (Cited on page 69.)

[25] 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, Department of Information and Computing Sciences, Utrecht University, 2008. (Cited on page 140.)

[26] 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. (Cited on pages ii, v, 6, 13, 18, 139, 141, 150, 151 and 183.)

[27] Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos. (Meta) kernelization. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS '09)*, pages 629–638. IEEE, 2009. (Cited on pages 69 and 138.)

[28] Hans L. Bodlaender, Stéphan Thomassé, and Anders Yeo. Kernel bounds for disjoint cycles and disjoint paths. In *Proceedings of the 17th Annual European Symposium on Algorithms (ESA '09)*, volume 5757 of *Lecture Notes in Computer Science*, pages 635–646. Springer, 2009. (Cited on page 20.)

[29] Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Cross-Composition: A New Technique for Kernelization Lower Bounds. In *Proceedings of the 28th International Symposium on Theoretical Aspects of Computer Science (STACS 2011)*, volume 9 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 165–176. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2011. (Cited on pages 6, 18, 19 and 137.)

[30] Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Preprocessing for treewidth: A combinatorial analysis through kernelization. In *Proceedings of the 38th International Colloquium on Automata, Languages and Programming (ICALP '11), Part I*, volume 6755 of *Lecture Notes in Computer Science*, pages 437–448. Springer, 2011. (Cited on pages 20, 23

and 161.)

[31] Francis T. Boesch, Charles Suffel, and Ralph Tindell. The spanning subgraphs of Eulerian graphs. *Journal of Graph Theory*, 1(1):79–84, 1977. (Cited on pages 111, 121 and 123.)

[32] Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph Classes: A Survey*, volume 3 of *SIAM Monographs on Discrete Mathematics and Applications*. SIAM, 1999. (Cited on page 8.)

[33] Kevin Burrage, Vladimir Estivill-Castro, Michael R. Fellows, Michael A. Langston, Shev Mac, and Frances A. Rosamond. The undirected feedback vertex set problem has a poly(k) kernel. In *Proceedings of the 2nd International Workshop on Parameterized and Exact Computation (IWPEC '06)*, volume 4169 of *Lecture Notes in Computer Science*, pages 192–202. Springer, 2006. (Cited on page 69.)

[34] Pablo Burzyn, Flavia Bonomo, and Guillermo Durán. NP-completeness results for edge modification problems. *Discrete Applied Mathematics*, 154(13):1824–1844, 2006. (Cited on page 108.)

[35] Jonathan F. Buss and Judy Goldsmith. Nondeterminism within *P*. *SIAM Journal on Computing*, 22(3):560–572, 1993. (Cited on pages 17 and 164.)

[36] Edgar Alberto Cabral, Michel Gendreau, Gianpaolo Ghiani, and Gilbert Laporte. Solving the hierarchical Chinese postman problem as a rural postman problem. *European Journal of Operational Research*, 155(1):44–50, 2004. (Cited on page 111.)

[37] Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58(4):171–176, 1996. (Cited on page 14.)

[38] Leizhen Cai and Boting Yang. Parameterized complexity of even/odd subgraph problems. *Journal of Discrete Algorithms*, 9(3):231–240, 2011. (Cited on page 112.)

[39] Liming Cai, Jianer Chen, Rodney G. Downey, and Michael R. Fellows. Advice classes of parameterized tractability. *Annals of Pure and Applied Logic*, 84(1):119–138, 1997. (Cited on pages 16 and 139.)

[40] Katarína Cechlárová and Tamás Fleiner. Housing markets through graphs. *Algorithmica*, 58 (1):19–33, 2010. (Cited on page 112.)

[41] Katarína Cechlárová and Ildikó Schlotter. Computing the deficiency of housing markets with duplicate houses. In *Proceedings of the 5th International Workshop on Parameterized and Exact Computation (IPEC '10)*, volume 6478 of *Lecture Notes in Computer Science*, pages 72–83. Springer, 2010. (Cited on page 112.)

[42] Pierre Charbit, Stéphan Thomassé, and Anders Yeo. The minimum feedback arc set problem is NP-hard for tournaments. *Combinatorics, Probability and Computing*, 16:1–4, 2007. (Cited on page 109.)

[43] Amitabh Chaudhary, Danny Z. Chen, Kevin Whitton, Michael T. Niemier, and Ramprasad Ravichandran. Eliminating wire crossings for molecular quantum-dot cellular automata implementation. In *Proceedings of the 2005 International Conference on Computer-Aided Design (ICCAD 2005)*, pages 565–571. IEEE Computer Society, 2005. (Cited on pages 62 and 63.)

[44] Amitabh Chaudhary, Danny Z. Chen, Rudolf Fleischer, Xiaobo Sharon Hu, Jian Li, Michael T. Niemier, Zhiyi Xie, and Hong Zhu. Approximating the maximum sharing problem. In *Proceedings of the 10th Workshop on Algorithms and Data Structures (WADS'07)*, volume 4619 of *Lecture Notes in Computer Science*, pages 52–63. Springer, 2007. (Cited on page 62.)

[45] Danny Z. Chen, Rudolf Fleischer, Jian Li, Zhiyi Xie, and Hong Zhu. On approximating the maximum simple sharing problem. In *Proceedings of the 17th International Symposium on Algorithms and Computation (ISAAC'06)*, volume 4288 of *Lecture Notes in Computer Science*, pages 547–556. Springer, 2006. (Cited on page 62.)

[46] Jianer Chen, Henning Fernau, Iyad A. Kanj, and Ge Xia. Parametric duality and kernelization: Lower bounds and upper bounds on kernel size. *SIAM Journal on Computing*, 37(4):1077–

1106, 2007. (Cited on pages 70, 71, 72 and 162.)

[47] Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theoretical Computer Science*, 411(40-42):3736–3756, 2010. (Cited on page 3.)

[48] Yijia Chen, Jörg Flum, and Moritz Müller. Lower bounds for kernelizations and other preprocessing procedures. *Theory of Computing Systems*, 48(4):803–839, 2011. (Cited on page 16.)

[49] Benny Chor, Michael R. Fellows, and David W. Juedes. Linear kernels in linear time, or how to save k colors in $O(n^2)$ steps. In *Proceedings of the 30th International Workshop on Graph-Theoretic Concepts in Computer Science (WG '04)*, volume 3353 of *Lecture Notes in Computer Science*, pages 257–269. Springer, 2004. (Cited on page 106.)

[50] Nicos Christofides. Worst case analysis of a new heuristic for the traveling salesman problem. Management Science Research Rept. 388, Carnegie-Mellon University, Pittsburgh, PA, 1976. (Cited on page 112.)

[51] Joseph Chuang-Chieh Lin. Computing branching numbers, 2009. URL `idv.sinica.edu.tw/josephcclin/paper/bn.pdf`. Source code available from `http://www.cs.ccu.edu.tw/~lincc/Program/br.c`. (Cited on page 58.)

[52] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing (STOC '71)*, pages 151–158, 1971. (Cited on page 140.)

[53] Bruno Courcelle. Clique-width of countable graphs: a compactness property. *Discrete Mathematics*, 276(1-3):127–148, 2004. (Cited on page 62.)

[54] Robert Crowston, Gregory Gutin, Mark Jones, and Anders Yeo. Parameterized Eulerian strong component arc deletion problem on tournaments. *Information Processing Letters*, 112 (6):249–251, 2012. (Cited on page 113.)

[55] Marek Cygan, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Ildikó Schlotter. Parameterized complexity of Eulerian deletion problems. In *Proceedings of the 37th International Workshop on Graph-Theoretic Concepts in Computer Science (WG '11)*, volume 6986 of *Lecture Notes in Computer Science*, pages 131–142. Springer, 2011. (Cited on page 112.)

[56] Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. An improved fpt algorithm and a quadratic kernel for pathwidth one vertex deletion. *Algorithmica*, 64(1):170–188, 2012. (Cited on page 27.)

[57] Martin Davis. What is...Turing reducibility? *Notices of the American Mathematical Society*, 53(10):1218–1219, 2006. (Cited on page 140.)

[58] Holger Dell and Dieter van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC 2010)*, pages 251–260. ACM, 2010. (Cited on pages 6, 141, 150, 151 and 160.)

[59] Erik D. Demaine, Fedor V. Fomin, Mohammad Taghi 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. (Cited on page 138.)

[60] Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, 2005. (Cited on pages 21 and 153.)

[61] Michael Dom, Daniel Lokshtanov, and Saket Saurabh. Incompressibility through colors and IDs. In *Proceedings of the 36th International Colloquium on Automata, Languages, and Programming (ICALP '09)*, volume 5555 of *Lecture Notes in Computer Science*, pages 378–389. Springer, 2009. (Cited on pages 19 and 165.)

[62] Michael Dom, Jiong Guo, Falk Hüffner, Rolf Niedermeier, and Anke Truss. Fixed-parameter tractability results for feedback set problems in tournaments. *Journal of Discrete Algorithms*, 8(1):76–86, 2010. (Cited on page 69.)

[63] Michael Dom, Jiong Guo, and Rolf Niedermeier. Approximation and fixed-parameter

algorithms for consecutive ones submatrix problems. *Journal of Computer and System Sciences*, 76(3-4):204–221, 2010. (Cited on page 27.)

[64] Frederic Dorn, Fedor V. Fomin, and Dimitrios M. Thilikos. Subexponential parameterized algorithms. *Computer Science Review*, 2(1):29–39, 2008. (Cited on page 71.)

[65] Frederic Dorn, Hannes Moser, Rolf Niedermeier, and Mathias Weller. Efficient algorithms for Eulerian extension and rural postman. *SIAM Journal on Discrete Mathematics*, 27(1): 75–94, 2013. (Cited on page viii.)

[66] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer, 1999. (Cited on pages 12, 13 and 144.)

[67] Moshe Dror, editor. *Arc Routing: Theory, Solutions, and Applications*. Kluwer Academic Publishers, 2000. (Cited on pages 108, 110, 111 and 138.)

[68] Andrew Drucker. New limits to classical and quantum instance compression. In *Proceedings of the 53rd Annual Symposium on Foundations of Computer Science (FOCS'12)*, pages 609–618. IEEE, 2012. (Cited on page 19.)

[69] Vida Dujmović, Michael R. Fellows, Michael T. Hallett, Matthew Kitching, Giuseppe Liotta, Catherine McCartin, Naomi Nishimura, Prabhakar Ragde, Frances A. Rosamond, Matthew Suderman, Sue Whitesides, and David R. Wood. A fixed-parameter approach to 2-layer planarization. *Algorithmica*, 45(2):159–182, 2006. (Cited on pages 24, 26, 27, 31, 45, 46 and 60.)

[70] Vida Dujmovic, Michael R. Fellows, Matthew Kitching, Giuseppe Liotta, Catherine McCartin, Naomi Nishimura, Prabhakar Ragde, Frances A. Rosamond, Sue Whitesides, and David R. Wood. On the parameterized complexity of layered graph drawing. *Algorithmica*, 52(2): 267–292, 2008. (Cited on pages 26 and 63.)

[71] Peter Eades and Sue Whitesides. Drawing graphs in two layers. *Theoretical Computer Science*, 131(2):361–374, 1994. (Cited on pages 26 and 27.)

[72] Jack Edmonds. The Chinese postman problem. *Operations Research*, 13(1):73–77, 1965. (Cited on page 111.)

[73] Jack Edmonds and Ellis L. Johnson. Matching, Euler tours and the Chinese postman. *Mathematical Programming*, 5:88–124, 1973. (Cited on pages 118 and 119.)

[74] Hartmut Ehrig, Claudia Ermel, Falk Hüffner, Rolf Niedermeier, and Olga Runge. Confluence in data reduction: Bridging graph transformation and kernelization. In *How the World Computes*, volume 7318 of *Lecture Notes in Computer Science*, pages 193–202. Springer, 2012. (Cited on page 165.)

[75] Horst A. Eiselt, Michel Gendreau, and Gilbert Laporte. Arc routing problems part I: The Chinese postman problem. *Operations Research*, 43(2):231–242, 1995. (Cited on pages 110 and 111.)

[76] Horst A. Eiselt, Michel Gendreau, and Gilbert Laporte. Arc routing problems part II: The rural postman problem. *Operations Research*, 43(3):399–414, 1995. (Cited on pages 108, 110, 111 and 114.)

[77] David Eppstein. Nineteen proofs of Euler's formula: $V - E + F = 2$. URL `http://www.ics.uci.edu/~eppstein/junkyard/euler/`. (Cited on page 74.)

[78] David Eppstein, Maarten Löffler, and Darren Strash. Listing all maximal cliques in sparse graphs in near-optimal time. In *Proceedings of the 21st International Symposium on Algorithms and Computation (ISAAC 2010)*, volume 6506 of *Lecture Notes in Computer Science*, pages 403–414. Springer, 2010. (Cited on page 156.)

[79] Vladimir Estivill-Castro and Michael E. Houle. Fast randomized algorithms for robust estimation of location. In *Proceedings of the 1st International Workshop on Temporal, Spatial, and Spatio-Temporal Data Mining (TSDM '00)*, volume 2007 of *Lecture Notes in Computer Science*, pages 77–88. Springer, 2000. (Cited on page 69.)

[80] Shimon Even. *Graph Algorithms*. W. H. Freeman & Co., 1979. (Cited on pages 154 and 155.)

[81] Uriel Feige, Mohammad Taghi Hajiaghayi, and James R. Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM Journal on Computing*, 38(2): 629–657, 2008. (Cited on page 157.)

[82] Michael Fellows. Towards fully multivariate algorithmics: Some new results and directions in parameter ecology. In *Proceedings of the 20th International Workshop on Combinatorial Algorithms (IWOCA '09)*, volume 5874 of *Lecture Notes in Computer Science*, pages 2–10. Springer, 2009. (Cited on page 138.)

[83] Michael Fellows, Daniel Lokshtanov, Neeldhara Misra, Matthias Mnich, Frances Rosamond, and Saket Saurabh. The complexity ecology of parameters: An illustration using bounded max leaf number. *Theory of Computing Systems*, 45(4):822–848, 2009. (Cited on page 24.)

[84] Michael R. Fellows, Daniel Lokshtanov, Neeldhara Misra, Frances A. Rosamond, and Saket Saurabh. Graph layout problems parameterized by vertex cover. In *Proceedings of the 19th International Symposium on Algorithms and Computation (ISAAC'08)*, volume 5369 of *Lecture Notes in Computer Science*, pages 294–305. Springer, 2008. (Cited on page 24.)

[85] Michael R. Fellows, Ariel Kulik, Frances A. Rosamond, and Hadas Shachnai. Parameterized approximation via fidelity preserving transformations. In *Proceedings of the 39th International Colloquium on Automata, Languages, and Programming (ICALP 2012)*, volume 7391 of *Lecture Notes in Computer Science*, pages 351–362. Springer, 2012. (Cited on pages 6, 140 and 164.)

[86] Henning Fernau. Two-layer planarization: Improving on parameterized algorithmics. *Journal of Graph Algorithms and Applications*, 9(2):205–238, 2005. (Cited on page 26.)

[87] Henning Fernau and Daniel Raible. Searching trees: An essay. In *Proceedings of the 6th Annual Conference on Theory and Applications of Models of Computation (TAMC '09)*, volume 5532 of *Lecture Notes in Computer Science*, pages 59–70. Springer, 2009. (Cited on page 18.)

[88] Henning Fernau, Fedor V. Fomin, Daniel Lokshtanov, Daniel Raible, Saket Saurabh, and Yngve Villanger. Kernel(s) for problems with no kernel: On out-trees with many leaves. In *Proceedings of the 26th International Symposium on Theoretical Aspects of Computer Science (STACS 2009)*, volume 3 of *LIPIcs*, pages 421–432. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany, 2009. (Cited on pages 6, 140, 142 and 159.)

[89] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer, 2006. (Cited on pages 12 and 13.)

[90] Fedor V. Fomin and Petr A. Golovach. Parameterized complexity of connected even/odd subgraph problems. In *Proceedings of the 29th International Symposium on Theoretical Aspects of Computer Science (STACS '12)*, volume 14 of *Leibniz International Proceedings in Informatics*, pages 432–440. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2012. (Cited on page 112.)

[91] Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2010. (Cited on pages 15 and 18.)

[92] Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Bidimensionality and kernels. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '10)*, pages 503–510. ACM/SIAM, 2010. (Cited on page 69.)

[93] Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Linear kernels for (connected) dominating set on H-minor-free graphs. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'12)*, pages 82–93. Society for Industrial and Applied Mathematics, 2012. (Cited on page 69.)

[94] Lance Fortnow. The status of the P versus NP problem. *Communications of the ACM*, 52(9): 78–86, 2009. (Cited on page 11.)

[95] Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct PCPs for NP. *Journal of Computer and System Sciences*, 77(1):91–106, 2011. (Cited on

pages ii, v, 6, 18, 139, 141, 150, 151, 181 and 183.)

[96] Greg N. Frederickson. *Approximation Algorithms for NP-hard Routing Problems*. PhD thesis, Faculty of the Graduate School of the University of Maryland, 1977. (Cited on pages 112 and 138.)

[97] Greg N. Frederickson. Approximation algorithms for some postman problems. *Journal of the ACM*, 26(3):538–554, 1979. (Cited on pages viii, 111, 112, 114 and 138.)

[98] Vincent Froese. Combinatorial feature selection: Parameterized algorithms and complexity. Master's thesis, Technische Universität Berlin, 2012. (Cited on page 2.)

[99] Toshihiro Fujito. A unified approximation algorithm for node-deletion problems. *Discrete Applied Mathematics*, 86(2-3):213–231, 1998. (Cited on page 9.)

[100] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979. (Cited on pages 12 and 111.)

[101] William I. Gasarch. Guest column: The second P =?NP poll. *SIGACT News*, 43(2):53–77, 2012. (Cited on page 12.)

[102] Ronald L. Graham, Eugene L. Lawler, Jan K. Lenstra, and Alexander H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Mathematics*, 5:287–326, 1979. (Cited on page 110.)

[103] Sylvain Guillemot, Christophe Paul, and Anthony Perez. On the (non-)existence of polynomial kernels for P_l-free edge modification problems. In *Proceedings of the 5th International Workshop on Parameterized and Exact Computation (IPEC '10)*, volume 6478 of *Lecture Notes in Computer Science*, pages 147–157. Springer, 2010. (Cited on page 69.)

[104] Jiong Guo and Rolf Niedermeier. Linear problem kernels for NP-hard problems on planar graphs. In *Proceedings of the 34th International Colloquium on Automata, Languages, and Programming (ICALP '07)*, volume 4596 of *Lecture Notes in Computer Science*, pages 375–386. Springer, 2007. (Cited on pages 69 and 70.)

[105] Jiong Guo and Rolf Niedermeier. Invitation to data reduction and problem kernelization. *ACM SIGACT News*, 38(1):31–45, 2007. (Cited on pages 1, 16 and 69.)

[106] Jiong Guo, Falk Hüffner, and Rolf Niedermeier. A structural view on parameterizing problems: Distance from triviality. In *Proceedings of the 1st International Workshop on Parameterized and Exact Computation (IWPEC'04)*, volume 3162 of *Lecture Notes in Computer Science*, pages 162–173. Springer, 2004. (Cited on page 24.)

[107] Sushmita Gupta. Feedback arc set problem in bipartite tournaments. *Information Processing Letters*, 105(5):150–154, 2008. (Cited on page 109.)

[108] Michel Habib and Christophe Paul. A survey of the algorithmic aspects of modular decomposition. *Computer Science Review*, 4(1):41–59, 2010. (Cited on page 69.)

[109] Torben Hagerup. Linear-time kernelization for planar dominating set. In *Proceedings of the 6th International Symposium on Parameterized and Exact Computation (IPEC 2012)*, volume 7112 of *Lecture Notes in Computer Science*, pages 181–193. Springer, 2011. (Cited on pages viii and 70.)

[110] Torben Hagerup. Kernels for edge dominating set: Simpler or smaller. In *Proceedings of the 37th International Symposium on Mathematical Foundations of Computer Science (MFCS 2012)*, volume 7464 of *Lecture Notes in Computer Science*, pages 491–502. Springer, 2012. (Cited on page 70.)

[111] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. The Morgan Kaufmann Series In Data Management Systems. Morgan Kaufmann Publishers, 2001. (Cited on page 2.)

[112] Frank Harary and Allen J. Schwenk. A new crossing number for bipartite graphs. *Utilitas Math.*, 1:203–209, 1972. (Cited on page 25.)

[113] Danny Harnik and Moni Naor. On the compressibility of NP instances and cryptographic applications. *SIAM Journal on Computing*, 39(5):1667–1713, 2010. (Cited on pages 2 and 19.)

[114] Sepp Hartung, Christian Komusiewicz, and André Nichterlein. Parameterized algorithmics and computational experiments for finding 2-clubs. In *Proceedings of the 6th International Symposium on Parameterized and Exact Computation (IPEC 2012)*, volume 7112 of *Lecture Notes in Computer Science*, pages 231–241. Springer, 2012. (Cited on page 161.)

[115] Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210, 1962. (Cited on pages 15, 16 and 112.)

[116] Danny Hermelin and Xi Wu. Weak compositions and their applications to polynomial lower bounds for kernelization. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'12)*, pages 104–113. Society for Industrial and Applied Mathematics, 2012. (Cited on pages 6 and 160.)

[117] Danny Hermelin, Stefan Kratsch, Karolina Soltys, Magnus Wahlström, and Xi Wu. Hierarchies of inefficient kernelizability. *CoRR*, abs/1110.0976, 2011. (Cited on pages 19, 142, 150, 153 and 159.)

[118] Danny Hermelin, Matthias Mnich, Erik Jan van Leeuwen, and Gerhard J. Woeginger. Domination when the stars are out. In *Proceedings of the 38th International Colloquium on Automata, Languages and Programming (ICALP '11), Part I*, volume 6755 of *Lecture Notes in Computer Science*, pages 462–473. Springer, 2011. (Cited on page 106.)

[119] Wiebke Höhn, Tobias Jacobs, and Nicole Megow. On Eulerian extension problems and their application to sequencing problems. Technical Report 008, Combinatorial Optimization and Graph Algorithms, TU Berlin, 2009. (Cited on page viii.)

[120] Wiebke Höhn, Tobias Jacobs, and Nicole Megow. On eulerian extensions and their application to no-wait flowshop scheduling. *Journal of Scheduling*, 15(3):295–309, 2012. (Cited on pages 109, 110, 111, 113, 114 and 118.)

[121] Falk Hüffner, Nadja Betzler, and Rolf Niedermeier. Optimal edge deletions for signed graph balancing. In *Proceedings of the 6th Workshop on Experimental Algorithms (WEA '07)*, number 4525 in LNCS, pages 297–310. Springer, June 2007. (Cited on page 69.)

[122] Bart Jansen. Kernelization for a hierarchy of structural parameters, 2011. Invited talk at the 3rd Workshop on Kernelization (Worker 2011). Slides available at `http://www.kr.tuwien.ac.at/drm/ordyniak/worker/slides/Bart-Jansen.pptx`. (Cited on page 22.)

[123] Bart Jansen and Stefan Kratsch. Data reduction for graph coloring problems. In *Fundamentals of Computation Theory*, volume 6914 of *Lecture Notes in Computer Science*, pages 90–101. Springer, 2011. (Cited on pages 23 and 161.)

[124] Bart Jansen and Stefan Kratsch. On polynomial kernels for structural parameterizations of odd cycle transversal. In *Proceedings of the 6th International Symposium on Parameterized and Exact Computation (IPEC 2012)*, volume 7112 of *Lecture Notes in Computer Science*, pages 132–144. Springer, 2011. (Cited on pages 23 and 161.)

[125] Bart M. P. Jansen and Hans L. Bodlaender. Vertex cover kernelization revisited: Upper and lower bounds for a refined parameter. In *28th International Symposium on Theoretical Aspects of Computer Science (STACS'11)*, volume 9 of *LIPIcs*, pages 177–188. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011. (Cited on pages 23, 24 and 161.)

[126] Klaus Jansen. An approximation algorithm for the general routing problem. *Information Processing Letters*, 41(6):333–339, 1992. (Cited on page 112.)

[127] Haitao Jiang and Binhai Zhu. Weak kernels. *Electronic Colloquium on Computational Complexity (ECCC)*, 17:5, 2010. (Cited on pages 6 and 140.)

[128] Haitao Jiang, Binhai Zhu, and Daming Zhu. Algorithms for sorting unsigned linear genomes by the DCJ operations. *Bioinformatics*, 27(3):311–316, 2011. (Cited on pages 6 and 140.)

[129] Minghui Jiang. On the parameterized complexity of some optimization problems related to multiple-interval graphs. *Theoretical Computer Science*, 411:4253–4262, 2010. (Cited on page 5.)

[130] Michael Jünger and Petra Mutzel. 2-layer straightline crossing minimization: Performance of exact and heuristic algorithms. *Journal of Graph Algorithms and Applications*, 1, 1997. (Cited on page 60.)

[131] Dieter Jungnickel. *Graphs, Networks and Algorithms*. Springer, 3rd edition, 2007. (Cited on page 153.)

[132] 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. *CoRR*, abs/1207.0835, 2012. (Cited on pages 69 and 138.)

[133] Pierre-Yves Koenig, Guy Melancon, Charles Bohan, and Berengere Gautier. Combining dagmaps and Sugiyama layout for the navigation of hierarchical data. In *Proceedings of the 11th International Conference on Information Visualisation (IV'07)*, pages 447–452. IEEE Computer Society, 2007. (Cited on page 26.)

[134] Christian Komusiewicz and Rolf Niedermeier. New races in parameterized algorithmics. In *Proceedings of the 37th International Symposium on Mathematical Foundations of Computer Science (MFCS 2012)*, volume 7464 of *Lecture Notes in Computer Science*, pages 19–30. Springer, 2012. (Cited on pages 5 and 136.)

[135] Bernhard H. Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*, volume 21 of *Algorithms and Combinatorics*. Springer, 5th edition, 2012. (Cited on page 119.)

[136] Stefan Kratsch. Co-nondeterminism in compositions: a kernelization lower bound for a Ramsey-type problem. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'12)*, pages 114–122. Society for Industrial and Applied Mathematics, 2012. (Cited on pages 6 and 151.)

[137] Stefan Kratsch and Pascal Schweitzer. Isomorphism for graphs of bounded feedback vertex set number. In *Proceedings of the 12th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT'10)*, volume 6139 of *Lecture Notes in Computer Science*, pages 81–92. Springer, 2010. (Cited on page 24.)

[138] Gilbert Laporte and Ibrahim H. Osman. Routing problems: A bibliography. *Annals of Operations Research*, 61:227–262, 1995. (Cited on page 111.)

[139] Thomas Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*. John Wiley & Sons, Inc., New York, NY, USA, 1990. (Cited on page 26.)

[140] Jan K. Lenstra and Alexander H. G. Rinnooy Kan. On general routing problems. *Networks*, 6 (3):273–280, 1976. (Cited on pages 110, 111, 112, 113, 118 and 123.)

[141] Linda Lesniak and Ortrud R. Oellermann. An Eulerian exposition. *Journal of Graph Theory*, 10(3):277–297, 1986. (Cited on pages 111 and 121.)

[142] John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980. (Cited on page 9.)

[143] Daniel Lokshtanov. *New Methods in Parameterized Algorithms and Complexity*. PhD thesis, University of Bergen, Norway, April 2009. (Cited on pages 6 and 140.)

[144] Daniel Lokshtanov. Generalization and specialization of kernelization, 2011. Invited talk at the 3rd Workshop on Kernelization (Worker 2011). Slides available at `http://www.kr.tuwien.ac.at/drm/ordyniak/worker/slides/Daniel-Lokshtanov.pptx`. (Cited on page 164.)

[145] Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Kernelization - preprocessing with a guarantee. In *The Multivariate Algorithmic Revolution and Beyond - Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday*, volume 7370 of *Lecture Notes in Computer Science*, pages 129–161. Springer, 2012. (Cited on pages 1, 16 and 20.)

[146] Meena Mahajan and Venkatesh Raman. Parameterizing above guaranteed values: MaxSat and MaxCut. *Journal of Algorithms*, 31(2):335–354, 1999. (Cited on pages 4 and 116.)

[147] Meena Mahajan, Venkatesh Raman, and Somnath Sikdar. Parameterizing above or below

guaranteed values. *Journal of Computer and System Sciences*, 75(2):137–153, 2009. (Cited on page 116.)

[148] John Markoff. Prizes aside, the P-NP puzzler has consequences. The New York Times, October 7th, 2009. Available online at http://www.nytimes.com/2009/10/08/science/Wpolynom.html. (Cited on page 11.)

[149] Dániel Marx. Parameterized graph separation problems. *Theoretical Computer Science*, 351 (3):394–406, 2006. (Cited on page 157.)

[150] Dániel Marx. What's next? future directions in parameterized complexity. In *The Multivariate Algorithmic Revolution and Beyond*, volume 7370 of *Lecture Notes in Computer Science*, pages 469–496. Springer, 2012. (Cited on page 162.)

[151] Manuel Sorge Mathias Weller, Rolf Niedermeier. Novel directions in data reduction for NP-hard problems: Truth-table kernelizations, 2013. Manuscript. (Cited on page 156.)

[152] Ross M. McConnell and Fabien de Montgolfier. Linear-time modular decomposition of directed graphs. *Discrete Applied Mathematics*, 145(2):198–209, 2005. (Cited on page 69.)

[153] Kwan Mei-Ko. Graphic programming using odd or even points. *Chinese Mathematics*, 1: 273–277, 1962. (Cited on page 111.)

[154] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography*. CRC Press Series on Discrete Mathematics and Its Applications. Taylor & Francis, 1996. (Cited on page 2.)

[155] Neeldhara Misra, Venkatesh Raman, and Saket Saurabh. Lower bounds on kernelization. *Discrete Optimization*, 8(1):110–128, 2011. (Cited on page 6.)

[156] Petra Mutzel. An alternative method to crossing minimization on hierarchical graphs. *SIAM Journal on Optimization*, 11(4):1065–1080, 2001. (Cited on pages vii, 26 and 59.)

[157] N.S. Narayanaswamy, Venkatesh Raman, M.S. Ramanujan, and Saket Saurabh. LP can be a cure for parameterized problems. In *29th International Symposium on Theoretical Aspects of Computer Science (STACS 2012)*, volume 14 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 338–349. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2012. (Cited on page 116.)

[158] Assaf Natanzon, Ron Shamir, and Roded Sharan. Complexity classification of some edge modification problems. *Discrete Applied Mathematics*, 113:109–128, 2001. (Cited on page 108.)

[159] André Nichterlein, Rolf Niedermeier, Johannes Uhlmann, and Mathias Weller. On tractable cases of target set selection. *Social Network Analysis and Mining*, pages 1–24, 2012. (Cited on pages 23, 24 and 161.)

[160] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Number 31 in Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, 2006. (Cited on pages 12, 13 and 161.)

[161] Rolf Niedermeier. Reflections on multivariate algorithmics and problem parameterization. In *Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science (STACS '10)*, volume 5 of *Leibniz International Proceedings in Informatics*, pages 17–32. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2010. (Cited on pages 138 and 161.)

[162] H. N. de Ridder. Information system on graph classes and their inclusions (ISGCI), Accessed July 2012. URL `http://www.graphclasses.org/`. (Cited on page 8.)

[163] John J. O'Connor and Edmund F. Robertson. Babylonian and Egyptian mathematics, MacTutor history of mathematics, University of St. Andrews, 2001. URL `http://www-history.mcs.st-and.ac.uk/HistTopics/Babylonian_and_Egyptian.html`. Accessed July 2012. (Cited on page 1.)

[164] Clifford S. Orloff. On general routing problems: Comments. *Networks*, 6(3):281–284, 1976. (Cited on pages viii, 112 and 138.)

[165] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994. (Cited on

pages 9, 10, 11, 143, 181 and 182.)

[166] Wen Lea Pearn, Arjang Assad, and Bruce L. Golden. Transforming arc routing into node routing problems. *Computers and Operations Research*, 14(4):285–288, 1987. (Cited on page 112.)

[167] Andrzej Proskurowski and Jan Arne Telle. Classes of graphs with restricted interval models. *Discrete Mathematics and Theoretical Computer Science*, 3(4):167–176, 1999. (Cited on page 27.)

[168] Fábio Protti, Maise Dantas da Silva, and Jayme L. Szwarcfiter. Applying modular decomposition to parameterized cluster editing problems. *Theory of Computing Systems*, 44(1):91–104, 2009. (Cited on page 70.)

[169] Elaine Rich. *Automata, Computability and Complexity: Theory and Applications*. Prentice Hall, 2008. (Cited on page 140.)

[170] RSA Laboratories. PKCS #1 v2.1: RSA encryption standard, June 2002. URL `ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf`. (Cited on page 2.)

[171] Stan Salvador and Philip Chan. Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis*, 11(5):561–580, 2007. (Cited on page 69.)

[172] Alexander Schäfer, Christian Komusiewicz, Hannes Moser, and Rolf Niedermeier. Parameterized computational complexity of finding small-diameter subgraphs. *Optimization Letters*, 6(5):883–891, 2012. (Cited on pages 142 and 159.)

[173] Richard Schroeppel and Adi Shamir. A $T = O(2^{n/2})$, $S = O(2^{n/4})$ algorithm for certain NP-complete problems. *SIAM Journal on Computing*, 10(3):456–464, 1981. (Cited on page 107.)

[174] Farhad Shahrokhi, Ondrej Sýkora, László A. Székely, and Imrich Vrto. On bipartite drawings and the linear arrangement problem. *SIAM Journal on Computing*, 30(6):1773–1789, 2001. (Cited on pages 24, 26 and 63.)

[175] Manuel Sorge, René van Bevern, Rolf Niedermeier, and Mathias Weller. A new view on rural postman based on Eulerian extension and matching. In *Proceedings of the 22nd International Workshop on Combinatorial Algorithms (IWOCA'11)*, volume 7056 of *Lecture Notes in Computer Science*, pages 310–323. Springer, 2011. (Cited on page viii.)

[176] Manuel Sorge, René van Bevern, Rolf Niedermeier, and Mathias Weller. From few components to an Eulerian graph by adding arcs. In *Proceedings of the 37th International Workshop on Graph-Theoretic Concepts in Computer Science (WG '11)*, volume 6986 of *Lecture Notes in Computer Science*, pages 307–318. Springer, 2011. (Cited on page viii.)

[177] Manuel Sorge, Hannes Moser, Rolf Niedermeier, and Mathias Weller. Exploiting a hypergraph model for finding Golomb rulers. In *Proceedings of the 2nd International Symposium on Combinatorial Optimization (ISCO 2012)*, volume 7422 of *Lecture Notes in Computer Science*, pages 368–379. Springer, 2012. Long version submitted to *Acta Informatica*. (Cited on page 106.)

[178] Manuel Sorge, René van Bevern, Rolf Niedermeier, and Mathias Weller. A new view on rural postman based on Eulerian extension and matching. *Journal of Discrete Algorithms*, 16: 12–33, 2012. (Cited on pages viii and 137.)

[179] Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM*, 51(3):385–463, 2004. (Cited on page 12.)

[180] Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis: An attempt to explain the behavior of algorithms in practice. *Communications of the ACM*, 52(10):76–84, 2009. (Cited on page 12.)

[181] Larry J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1): 1–22, 1976. (Cited on page 182.)

[182] Matthew Suderman. *Layered Graph Drawing*. PhD thesis, School of Computer Science,

McGill University Montréal, 2005. (Cited on pages i, iv, vii, 24, 26, 27, 45, 48, 49, 50, 51, 52, 53, 57, 59, 60, 61, 62 and 64.)

[183] Matthew Suderman and Sue Whitesides. Experiments with the fixed-parameter approach for two-layer planarization. *Journal of Graph Algorithms and Applications*, 9(1):149–163, 2005. (Cited on pages vii and 59.)

[184] Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiko Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man and Cybernetics*, 11(2): 109–125, 1981. (Cited on page 26.)

[185] Jinsong Tan and Louxin Zhang. The consecutive ones submatrix problem for sparse matrices. *Algorithmica*, 48(3):287–299, 2007. (Cited on page 27.)

[186] Robert Endre Tarjan. A note on finding the bridges of a graph. *Information Processing Letters*, 2(6):160–161, 1974. (Cited on page 58.)

[187] Shang-Hua Teng. Algorithmic primitives for network analysis: Through the lens of the laplacian paradigm, 2012. Talk at the *2012 SIAM International Conference on Data Mining (SDM '12)*. (Cited on page 69.)

[188] Stéphan Thomassé. A $4k^2$ kernel for feedback vertex set. *ACM Transactions on Algorithms*, 6(3):32:1–32:8, 2010. (Cited on page 69.)

[189] Vassilis Tsiaras, Sofia Triantafilou, and Ioannis G. Tollis. DAGmaps: Space filling visualization of directed acyclic graphs. *Journal of Graph Algorithms and Applications*, 13(3): 319–347, 2009. (Cited on pages 62 and 63.)

[190] A. M. Turing. Systems of logic based on ordinals. *Proceedings of the London Mathematical Society*, 45:161–228, 1939. (Cited on page 140.)

[191] Johannes Uhlmann and Mathias Weller. Two-layer planarization parameterized by feedback edge set. In *Proceedings of the 7th Annual Conference on Theory and Applications of Models of Computation (TAMC '10)*, volume 6108 of *LNCS*, pages 431–442. Springer, 2010. (Cited on page vii.)

[192] Vijay V. Vazirani. *Approximation Algorithms*. Springer, 2001. (Cited on page 12.)

[193] Magnus Walström. The AND-conjecture may be neccessary. *Parameterized Complexity Newsletter*, page 3, November 2011. (Cited on page 151.)

[194] Jianxin Wang, Yongjie Yang, Jiong Guo, and Jianer Chen. Linear problem kernels for planar graph problems with small distance property. In *Proceedings of the 36th International Symposium on Mathematical Foundations of Computer Science (MFCS '11)*, volume 6907 of *Lecture Notes in Computer Science*, pages 592–603. Springer, 2011. (Cited on pages viii and 70.)

[195] Michael S. Waterman and Jerrold R. Griggs. Interval graphs and maps of DNA. *Bulletin of Mathematical Biology*, 48(2):189–195, 1986. (Cited on page 26.)

[196] Karsten Weihe. Covering trains by stations or the power of data reduction. In *Proceedings of the 1st Workshop on Algorithms and Experiments (ALEX '98)*, pages 1–8, 1998. (Cited on page 2.)

[197] Karsten Weihe. On the differences between "practical" and "applied". In *Proceedings of the 4th International Workshop on Algorithm Engineering (WAE '00)*, volume 1982 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2000. (Cited on page 2.)

[198] Mathias Weller, Christian Komusiewicz, Rolf Niedermeier, and Johannes Uhlmann. On making directed graphs transitive. *Journal of Computer and System Sciences*, 78(2):559–574, 2012. (Cited on page 109.)

[199] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011. (Cited on page 12.)

[200] Gerhard J. Woeginger. Exact algorithms for NP-hard problems: A survey. In *Proceedings of the 5th International Workshop on Combinatorial Optimization (IWCO 2003) - Eureka, You Shrink!, Papers Dedicated to Jack Edmonds*, volume 2570 of *Lecture Notes in Computer*

Science, pages 185–208. Springer, 2003. (Cited on page 107.)

[201] Mingyu Xiao, Ton Kloks, and Sheung-Hung Poon. New parameterized algorithms for the edge dominating set problem. In *Proceedings of the 36th International Symposium on Mathematical Foundations of Computer Science (MFCS 2011)*, volume 6907 of *Lecture Notes in Computer Science*, pages 604–615. Springer, 2011. (Cited on page 70.)

[202] Mihalis Yannakakis. Node-deletion problems on bipartite graphs. *SIAM Journal on Computing*, 10(2):310–327, 1981. (Cited on page 9.)

Appendix A

Plausibility of NP $\subseteq$ coNP/poly

Our (and practically all known) kernelization lower bounds (see Section 4.6) are based on the assumption that NP $\not\subseteq$ coNP/poly. As shown by Fortnow and Santhanam [95], NP $\subseteq$ coNP/poly implies a collapse of the polynomial hierarchy PH to its third level.

One might ask "Why is a collapse of the polynomial hierarchy unlikely?", so we are going to delve into this question, trying to visualize implications of NP $\subseteq$ coNP/poly. We like to think of the polynomial hierarchy in terms of complete problems. To this end, we call a boolean formula φ *quantified* if all variables occurring in φ are quantified by a $\forall$ or $\exists$ quantifier with their usual meaning. Such a formula evaluates to true or false and can be seen as the following two-player game [165]. Player "$\exists$" and player "$\forall$" get the unquantified boolean formula. Then, the variables are considered in order of their quantification in the quantified boolean formula and, depending on the type of quantification, the respective player may choose a value for this variable. For example, if the quantified boolean formula starts with "$\exists_x \forall_y$", then player "$\exists$" may choose the value of x and then player "$\forall$" may choose a value for y. Then, the quantified boolean formula evaluates to true if and only if player "$\exists$" has a strategy that always wins, no matter what player "$\forall$" does. We call such a strategy *solution* for the quantified boolean formula.

True Quantified Boolean Formula
Input: A quantified boolean formula φ.
Question: Does φ evaluate to true?

Input:	**Solution:**
$\underset{x_1}{\forall}\ \underset{x_2,x_3}{\exists}\ \underset{x_4}{\forall}\ (x_1 \vee \neg x_2 \vee \neg x_4) \wedge (x_2 \vee x_3 \vee \neg x_4) \wedge (\neg x_1 \vee \neg x_3 \vee x_4)$	yes, "$\exists$" chooses $x_2 := x_1$ $x_3 := \neg x_1$

Note that permuting the quantifiers can create different instances. Therefore, it is meaningful to define α-True Quantified Boolean Formula, the set of all yes-instances of True Quantified Boolean Formula for which the formula φ contains exactly α quantifier alternations[38]. Likewise, we define $\exists$-α-True Quantified Boolean Formula as the set of all yes-instances of α-True Quantified Boolean Formula whose formula φ starts with "$\exists$". Now, we can define the polynomial hierarchy as follows [10, 165].

Definition A.1. *Let $\alpha \in \mathbb{N}$. The problem Σ_α^P is the set of all problems that can be many-one reduced to $\exists$-α-True Quantified Boolean Formula in polynomial time. Likewise, the problem Π_α^P is the set of all problems that can be many-one reduced to $\forall$-α-True Quantified Boolean Formula in polynomial time. Finally, the class of all problems in the polynomial hierarchy is* $\mathrm{PH} = \bigcup_i \Sigma_i^P = \bigcup_i \Pi_i^P$.

Notably, since the satisfyability problem for boolean formulae, SAT, is NP-complete and the tautology problem TAUT is coNP-complete, we can observe $\mathrm{NP} = \Sigma_1^P$ and $\mathrm{coNP} = \Pi_1^P$ [181, 165].
Now, in a world where the polynomial hierarchy collapses to its third level, it holds that $\mathrm{PH} = \Sigma_3^P = \Pi_3^P$. Thus, $\exists$-3-True Quantified Boolean Formula is complete for PH, implying that, given a quantified boolean formula φ with any number of quantifier alternations, we can transform φ into a formula with only three quantifier alternations in polynomial time. This does not seem much less likely than being able to eliminate *all* quantifiers in this way. This, however, is equal to P = NP, which contradicts the working hypothesis of this thesis.

[38] A *quantifier alternation* is a change in quantification type, that is, a transition from $\exists$ to $\forall$ or vice versa. Here, the end of the quantifier list counts as an alternation. In our notation of writing multiple variables below one quantifier, φ contains exactly α quantifiers.

In our hypothetical world in which PH = Σ_3^P, we have another problem. The problem True Quantified Boolean Formula stated above is complete for the complexity class PSPACE (the class of all problems solvable using only polynomial space). If now PH = PSPACE, then all problems solvable in polynomial space are expressible as a quantified boolean formula with at most three quantifier alternations, which seems unlikely to us. If PH ≠ PSPACE, then, although there is a polynomial-time many-one reduction R of α-True Quantified Boolean Formula to 3-True Quantified Boolean Formula for any fixed α, we cannot produce this reduction in polynomial time, since otherwise, we could reduce True Quantified Boolean Formula to 3-True Quantified Boolean Formula by producing R and running R on the instance.

For these reasons, NP ⊆ coNP/ poly seem unlikely and, thus, compositionality is likely to imply non-existence of polynomial-size kernels [26, 95].

Universitätsverlag der TU Berlin
ISBN 978-3-7983-2508-1 (Druckversion)
ISBN 978-3-7983-2509-8 (Onlineversion)

Technische Universität Berlin