Finding Optimal Solutions for Covering and Matching Problems

Dissertation

zur Erlangung des akademischen Grades
doctor rerum naturalium (Dr. rer. nat.)

vorgelegt dem Rat der Fakultät für Mathematik und Informatik
der Friedrich-Schiller-Universität Jena

von Dipl.-Inform. Hannes Moser
geboren am 18. 11. 1978 in München

Gutachter

- Prof. Dr. Rolf Niedermeier (Friedrich-Schiller-Universität Jena)
- Prof. Dr. Iyad Kanj (DePaul University, Chicago, U.S.A.)
- Prof. Dr. Dimitrios Thilikos (National and Kapodistrian University of Athens, Griechenland)

Tag der öffentlichen Verteidigung: 17. November 2009

Zusammenfassung

Diese Arbeit beschäftigt sich mit kombinatorischen Problemen, welche als Verallgemeinerungen der beiden klassischen Graphprobleme Vertex Cover und Maximum Matching aufgefasst werden können. Das Vertex Cover-Problem ist wie folgt definiert. Gegeben ein ungerichteter Graph, finde eine kleinstmögliche Knotenteilmenge, die jede Kante „abdeckt“, d.h. dass einer der beiden Endpunkte jeder Kante in der Knotenteilmenge liegt. Dieses Problem wird auch oft „Knotenüberdeckungsproblem“ genannt. Das Maximum Matching-Problem fragt nach einer größtmöglichen Kantenteilmenge in einem ungerichteten Graphen, so dass sich die gewählten Kanten keinen Endpunkt teilen. Dieses Problem sucht also nach einer möglichst großen Anzahl von Knotenpaaren, die durch eine Kante verbunden sind. In bipartiten Graphen wird dieses Problem auch oft „Heiratsproblem“ genannt.

Sowohl Vertex Cover als auch Maximum Matching haben eine lange Geschichte; diese Probleme wurden schon in den Anfangsjahren der Informatik untersucht und sind immer noch Gegenstand der aktuellen Forschung. Es gibt für beide Probleme viele Anwendungen, beispielsweise in der Bioinformatik, der Computer-Chemie oder auch in der Verkehrsplanung. Maximum Matching wird in unzähligen Anwendungen als Hilfsroutine zur Lösung anderer Aufgaben eingesetzt.

Ein fundamentaler Unterschied von Vertex Cover und Maximum Matching ist ihre algorithmische Komplexität: während Maximum Matching in Polynomzeit lösbar ist, was üblicherweise als effizient angesehen wird, ist Vertex Cover NP-schwer. Das bedeutet, dass es vermutlich keinen Polynomzeitalgorithmus für Vertex Cover gibt. Beide Probleme haben aber auch Gemeinsamkeiten, die sich in der beiden Problemen zugrundeliegenden Struktur „Kante“ widerspiegeln. Tatsächlich liegt diesen Gemeinsamkeiten auch ein berühmtes Ergebnis von König zugrunde, welches besagt, dass Vertex Cover und Maximum Matching in bipartiten Graphen äquivalent sind und damit Vertex Cover in bipartiten Graphen ebenfalls in Polynomzeit lösbar ist.

Die Probleme, die in dieser Arbeit untersucht werden, lassen sich grob in Knotenüberdeckungsprobleme und generalisierte Matching-Probleme unterteilen. Die Knotenüberdeckungsprobleme können wie folgt beschrieben werden. Für eine bestimmte Grapheigenschaft und gegebenem Graph, lösche möglichst wenige Knoten, so dass der resultierende Graph die besagte Grapheigenschaft besitzt. Vertex Cover entspricht diesem Problem mit Grapheigenschaft „kantenfrei“. Wir formulieren also die Knotenüberdeckungsprobleme als Knotenlöschungsprobleme, d.h. statt eine gewisse Struktur (wie z.B. Kanten) zu überdecken, sprechen wir von der Zerstörung der Struktur mit Hilfe von Knotenlöschungen. Die Matching-

Probleme können wie folgt beschrieben werden. Gegeben sei ein ungerichteter Graph.

- Finde eine größtmögliche Anzahl von Kopien eines fest vorgegebenen zusammenhängenden Graphen mit mindestens drei Knoten, die paarweise knotendisjunkt sind.

- Finde eine größtmögliche Anzahl von Kanten, die paarweise einen gewissen Mindestabstand haben müssen.

Diese Probleme sind alle NP-schwer, d.h. es kann vermutlich keine Polynomzeitalgorithmen zum Finden einer optimalen Lösung geben. Doch auch für NP-schwere Probleme können oft positive Resultate erzielt werden. Eine Möglichkeit sind Heuristiken, die oft bei bestimmten Instanzen in der Praxis sehr gute Lösungen liefern, deren Laufzeiten und/oder Lösungsgüten aber nicht bewiesen werden können. Eine weitere Herangehensweise sind Approximationsalgorithmen, welche in Polynomzeit eine Lösung finden, die nur um einen beweisbaren Faktor von einer optimalen Lösung abweicht. Die Probleme, die in dieser Arbeit behandelt werden, sind jedoch alle im besten Fall nur mit einem konstanten Faktor approximierbar, was in der Praxis oftmals nicht ausreichend ist. Eine weiterer Ansatz sind parametrisierte Algorithmen. Die Grundidee hierbei ist, die kombinatorische Komplexität eines NP-schweren Problems nicht nur bezüglich der Eingabegröße zu analysieren, sondern auch einen geschickt gewählten Parameter mit in die Analyse einfließen zu lassen. Ein Problem ist *festparameter-handhabbar* bezüglich eines Parameters k, wenn eine optimale Lösung einer Instanz der Größe n in Zeit $f(k) \cdot \text{poly}(n)$ gefunden werden kann. Die Idee dahinter ist, dass man für Instanzen mit kleinem Parameter gute Laufzeiten erhält, unabhängig von der Gesamtgröße der Eingabeinstanz. Diese Arbeit beschäftigt sich im Wesentlichen mit diesem parametrisierten Ansatz. Ein wichtiges Konzept, um zu zeigen, dass ein parametrisiertes Problem festparameter-handhabbar ist, sind *Problemkerne*. Ein Problemkern ist grob gesagt eine in Polynomzeit konstruierbare Instanz, die zur Eingabeinstanz äquivalent ist, aber deren Größe nur von dem Parameter (und nicht mehr von der Eingabegröße) abhängig ist. Aus einer optimalen Lösung für den Problemkern kann man dann eine optimale Lösung für die Eingabeinstanz berechnen. Im Folgenden werden die Ergebnisse dieser Arbeit im Überblick beschrieben.

Bounded-Degree Vertex Deletion. Hierbei handelt es sich um das Problem, einen gegebenen Graph durch Löschen von maximal k Knoten in einen Graph mit konstantem Maximalgrad zu überführen. Das wichtigste Ergebnis dazu ist ein Algorithmus, der in polynomieller Zeit zwei Knotenteilmengen berechnet, so dass man davon ausgehen kann, dass eine Knotenteilmenge immer in einer optimalen Lösung ist, dass die andere von der Suche nach einer optimalen Lösung ausgeschlossen werden kann, und dass eine optimale Lösung für den Rest des Graphen

eine bestimmte Mindestgröße besitzt. Dieses Ergebnis liefert einen Problemkern mit $O(k^{1+\epsilon})$ Knoten für ein beliebiges konstantes $\epsilon > 0$. Weitere Ergebnisse sind ein parametrisierter Algorithmus mit der Laufzeit $O((d+2)^k + kn)$ und ein paar schnellere Algorithmen für einen Spezialfall des Problems. Weiterhin wird gezeigt, dass das Problem vermutlich nicht mehr festparameter-handhabbar bezüglich Parameter k ist, wenn der Maximalgrad des Zielgraphen Teil der Eingabe ist.

Regular-Degree Vertex Deletion. Hierbei handelt es sich um das Problem, einen Graph durch Löschen von maximal k Knoten in einen regulären Graph zu überführen. Dieses Problem hat eine offensichtliche Ähnlichkeit zu Bounded-Degree Vertex Deletion, verhält sich aber aufgrund von hier nicht näher erläuterten Eigenschaften in wesentlichen Aspekten deutlich anders. Für dieses Problem wird gezeigt, dass es NP-schwer und festparameter-handhabbar bezüglich Parameter k ist. Das Hauptergebnis ist ein Problemkern mit $O(k^3)$ Knoten.

Knotenlöschungsprobleme und iterative Kompression. Iterative Kompression ist eine im Jahr 2004 entwickelte Technik, die auf struktureller Induktion und Kompression von Zwischenlösungen basiert. Diese Technik wurde in den letzten Jahren erfolgreich zur Lösung von einigen jahrelang offenen Problemen eingesetzt. Fast alle diese Probleme sind Knotenlöschungsprobleme. In beinahe allen Anwendungen dieser Technik wird eine Kompressionsaufgabe gelöst, welche bei einer gegebenen Zwischenlösung nach einer davon *disjunkten* kleineren Lösung fragt. Für eine große Klasse von Knotenlöschungsproblemen wird gezeigt, für welche Fälle die Kompressionsaufgabe NP-schwer ist und für welche Fälle sie in Polynomzeit gelöst werden kann. Für die in Polynomzeit lösbaren Fälle ergibt sich daraus auch direkt ein effizienter Festparameter-Algorithmus für das entsprechende Knotenlöschungsproblem, unter anderem für einen Spezialfall des oben beschriebenen Bounded-Degree Vertex Deletion.

Graph Packing. Bei diesem Problem geht es darum, in einem gegebenen Graph mindestens k knotendisjunkte Kopien eines festen Graphen H zu finden. Es wird zuerst ein Problemkern mit $O(k^2)$ Knoten für das Problem, mindestens k knotendisjunkte Dreiecke zu finden, gezeigt. Dieses Ergebnis verbessert ein bekanntes Resultat und hat den Vorteil, dass es auf beliebige zusammenhängende Graphen H erweitert werden kann, was zu einem Problemkern mit $O(k^{h-1})$ Knoten führt, wobei h die Anzahl der Knoten in H ist.

Induced Matching. Hier geht es darum, mindestens k Kanten zu finden, so dass sie paarweisen Abstand mindestens zwei haben. Bezüglich des Parameters k ist dieses Problem vermutlich nicht festparameter-handhabbar, daher wird die parametrisierte Komplexität dieses Problems auf speziellen Graphklassen untersucht. Untersucht werden unter anderem planare Graphen, Graphen mit beschränktem Knotengrad, bipartite Graphen und Graphen mit beschränkter

Baumweite. Das Hauptergebnis ist ein Problemkern der Größe $O(k)$ in planaren Graphen.

Maximum s-Plex. Bei diesem Problem geht es darum, in einem gegebenen Graph einen induzierten Teilgraph zu finden, in dem jeder Knoten zu maximal $s-1$ anderen nicht benachbart ist. Dabei soll die Anzahl der Knoten in dem Teilgraph größtmöglich sein. Für $s = 1$ entspricht dies dem klassischen Maximum Clique-Problem. Es ist offensichtlich, dass der Komplementärgraph eines Graphen, zu dem jeder Knoten zu maximal $s-1$ anderen nicht benachbart ist, Maximalknotengrad $s-1$ hat. Dies stellt einen direkten Bezug zu Bounded-Degree Vertex Deletion her. Es wird eine Implementierung basierend auf den theoretischen Ergebnissen für Bounded-Degree Vertex Deletion beschrieben. Dieser Ansatz ist in Experimenten für viele Graphen um Größenordnungen schneller als bisher bekannte Methoden.

Preface

This thesis summarizes a significant part of my studies and research on parameterized complexity, focusing on kernelization and fixed-parameter algorithms for covering and matching problems. My research was funded by the Deutsche Forschungsgemeinschaft (DFG) under the project "iterative compression for solving hard network problems" (ITKO, project number NI 369/5) from February 2006 until February 2008 and since February 2008 by the project "algorithms for generating quasi-regular structures in graphs" (AREG, project number NI 369/9). Before starting my research in Jena, I was also supported by the EC Research Training Network HPRN-CT-2002-00278 (COMBSTRU) while I was visiting Josep Díaz and Dimitrios Thilikos between November 2005 and February 2006 at the Universitat Politècnica de Catalunya, Barcelona, Spain.

I owe sincere thanks to my supervisor Rolf Niedermeier, for giving me the opportunity to work in his group and for the invaluable and time-intensive guidance in research, writing, and presentation skills. I also want to thank my colleagues Nadja Betzler, Michael Dom, Jiong Guo, Falk Hüffner, Christian Komusiewicz, Johannes Uhlmann, Matthias Weller, and Sebastian Wernicke for a pleasant working atmosphere and many interesting and fruitful discussions. In particular, Falk Hüffner, Christian Komusiewicz, and Jiong Guo are co-authors of several joint publications. Jiong Guo was involved in many projects I worked on, and I have learned a lot from him. I also want to thank my other co-authors Daniel Brügmann (Carl-Zeiss-Gymnasium Jena), René van Bevern (Universität Jena), Michael R. Fellows (University of Newcastle, Australia), Venkatesh Raman (Institute of Mathematical Sciences, Chennai, India), Somnath Sikdar (Institute of Mathematical Sciences, Chennai, India), Manuel Sorge (Universität Jena), and Dimitrios Thilikos (National and Kapodistrian University of Athens, Greece). Many thanks also to Robert Bredereck and Manuel Sorge for their support in implementing and performing experiments. In particular, Manuel Sorge contributed a lot of effort to the experimental results presented in this thesis. Parts of this thesis have also profited from discussions with Balabhaskar Balasundaram (Oklahoma State University, Stillwater, U.S.A.), Henning Fernau (Universität Trier, Germany), Danny Hermelin (University of Haifa, Israel), Matthias Hagen (Universität Weimar, Germany), Iyad Kanj (DePaul University, Chicago, U.S.A.), Dániel Marx (Budapest University of Technology and Economics, Hungary), Luke Mathieson (University of Durham, U.K.), Daniel Raible (Universität Trier, Germany), and Saket Saurabh (University of Bergen, Norway).

Last, but not least, I would like to thank my family for all their love and especially my parents who supported me in all my pursuits. And most of all I want to thank my beloved wife Sofía for living her life with me, her love and support, and for being a great mother to our lovely daughter Iris. Your encouragement, understanding, and patience in the times of preparing this thesis is so appreciated, and the thesis would not have such a nice cover without your help!

This thesis is structured into nine chapters. After a brief introduction in Chapter 1, all notation that is used throughout the thesis is described in Chapter 2. After that, I describe the central problems and give an overview on existing and new results in Chapter 3. Then, the subsequent two chapters deal with the covering problems (formulated as vertex deletion problems) Bounded-Degree Vertex Deletion (Chapter 4) and Regular-Degree Vertex Deletion (Chapter 5). Chapter 6 studies some central aspects of a technique called iterative compression with respect to a wide class of vertex deletion problems. The subsequent two chapters consider the Maximum Matching generalizations H-Packing (Chapter 7) and Induced Matching (Chapter 8). Finally, we report about an implementation and corresponding experimental results for Maximum s-Plex (Chapter 9), a problem that is closely related to Bounded-Degree Vertex Deletion. In the following, I briefly sketch my contributions.

Chapter 4 considers Bounded-Degree Vertex Deletion, the problem of deleting a minimum number of vertices from a given graph such that the resulting graph has bounded degree, where the degree bound is a constant. The main result is a generalization of a well-known local optimization algorithm for Vertex Cover to Bounded-Degree Vertex Deletion. This local optimization algorithm also gives an "almost linear" problem kernel for Bounded-Degree Vertex Deletion. The research was initiated by Jiong Guo and Rolf Niedermeier. The local optimization algorithm for Bounded-Degree Vertex Deletion (Section 4.3) was devised by Jiong Guo, Michael R. Fellows, Rolf Niedermeier, and me in various discussions. A first version of the local optimization algorithm was presented at the *26th International Symposium on Theoretical Aspects of Computer Science (STACS '09)* [FGMN09b]. Unfortunately, this version had a flaw, which was spotted by Jiří Sgall (Charles University, Prague, Czech Republic). However, the repaired version is based on exactly the same technique with only few a modifications. I developed the description of the (repaired) local optimization algorithm and the correctness proof. Moreover, some fixed-parameter algorithms are presented. The fixed-parameter algorithm for Bounded-Degree Vertex Deletion (Section 4.4.1) goes back to an idea by Jiong Guo. The improved fixed-parameter algorithm (Section 4.4.2) for a special case of Bounded-Degree Vertex Deletion is due to Rolf Niedermeier and me. The improved fixed-parameter algorithm was presented (among other results) at the *8th International Symposium on Experimental Algorithms (SEA '09)* [MNS09]. I found a fixed-parameter algorithm based on iterative compression (Section 4.5), where a linear-time subroutine that solves Maximum Matching on a restricted graph class to speed up the algorithm has been found by Manuel Sorge. It is also shown that Bounded-Degree Vertex Deletion is presumably not fixed-parameter tractable if the degree bound is part of the input. This hardness result (Section 4.6) was found in joint discussions together with Michael R. Fellows, Jiong Guo, and Rolf Niedermeier.

Chapter 5 considers Regular-Degree Vertex Deletion, the problem

of deleting a minimum number of vertices from a given graph such that the remaining graph is regular, where the degree of the regularity is constant. The Regular-Degree Vertex Deletion problem was proposed by Dimitrios Thilikos. An NP-completeness proof, a problem kernel, and a fixed-parameter algorithm are presented. The results were obtained by Dimitrios Thilikos and me in a number of discussions. The results were presented at the *2nd Algorithms and Complexity in Durham Workshop (ACiD '06)* [MT06] and appeared in the *Journal of Discrete Algorithms* [MT09].

Chapter 6 considers the iterative compression technique for vertex deletion problems; this study was initiated by me. We analyze the computational complexity of a generally occurring subtask when applying the iterative compression technique to vertex deletion problems. The proofs for the polynomial-time solvable cases are based on an iterative compression algorithm for Cluster Vertex Deletion; I came up with the idea of using matching techniques, the complete algorithm was then developed by Falk Hüffner. I devised the modifications for the other polynomial-time solvable cases. The hardness results were obtained in various discussions together with Michael F. Fellows, Jiong Guo, and Rolf Niedermeier. I discovered a relation to a framework by Yannakakis and also developed the hardness proofs for the cases in which the framework cannot be used. The results are presented at the *34th International Symposium on Mathematical Foundations of Computer Science (MFCS '09)* [FGMN09a]. Some parts of the introduction to iterative compression also appear in a survey on iterative compression [GMN09].

Chapter 7 considers H-Packing, the problem of packing a maximum number of vertex-disjoint copies of a fixed graph into a given graph. I initiated the study of H-Packing and present problem kernels for several cases; these problem kernels were also developed by me. The results were presented at the *35th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM '09)* [Mos09].

Chapter 6 considers the problem of packing a maximum number of edges that have pairwise distance at least two into a given graph. The study of the Induced Matching problem in Chapter 8 was initiated by Jiong Guo and Rolf Niedermeier. The fundamental parameterized complexity results for various graph classes were found by Somnath Sikdar and me in various discussions. The hardness reduction for bipartite graphs was found by Saket Saurabh and Daniel Lokshtanov. I devised the linear problem kernel and the dynamic programming result. The results were presented at the *1st International Frontiers of Algorithmics Workshop (FAW '07)* [MS07b] and appeared in *Discrete Applied Mathematics* [MS09b].

Finally, Chapter 9 considers the Maximum s-Plex problem, a generalization of Maximum Clique, and reports about implementation and experiments based on fixed-parameter techniques. I initiated the experimental work on Maximum s-Plex. I implemented and tested a first prototype, in later stages many of the

implementation work was done by Manuel Sorge. He also conducted many of the experiments. The results are presented at the *8th International Symposium on Experimental Algorithms (SEA '09)* [MNS09].

Jena, January 2010 Hannes Moser

Contents

Chapter 1

Introduction

This thesis deals with combinatorial problems that can be viewed as generalizations of the classical Vertex Cover and Maximum Matching problems.

Covering Problems. The Vertex Cover problem is a well-studied NP-hard combinatorial problem with many applications in different areas [BYE85, Hoc97]. One important example is conflict resolution in computational biochemistry [LBI+01]. After a series of experiments there might exist conflicts between the results of the experiments. For instance, when aligning DNA sequences, there usually exist conflicts between sequences in the sample (e.g., caused by sequencing errors). One possible way to resolve these conflicts is to remove sequences from the sample, until there are no conflicts anymore, and the goal is to remove as few sequences as possible. One can model this situation with a graph, where the vertices correspond to sequences and there is an edge between two vertices whenever there is a conflict between the corresponding sequences. Now, the task is to delete a minimum number of vertices from the graph such that the remaining graph contains no edges. In other words, the task is to find a smallest subset S of vertices such that every edge contains at least one vertex of S, that is, every edge is covered by at least one vertex in S.

> Vertex Cover
> **Instance:** An undirected graph.
> **Task:** Delete a minimum number of vertices from the graph such that there remain no edges.

Vertex Cover is very closely related to finding maximum-cardinality cliques. A clique is a subgraph in which there exists an edge between each pair of vertices. Finding large cliques is important in many practical applications and has been subject of the second DIMACS implementation challenge in 1995 [DIM95]. Recent papers describe applications in computational finance [BBP05, BBP06] and computational biochemistry and genomics [BW06, CLS+05]. Moreover, clique finding also plays a role in classical computer science fields such as the analysis

of web graphs for instance to identify web communities [FLG00] or "link farms" (for the purpose of spam deletion and analysis) [STKA07]. Finding maximum-cardinality cliques via an algorithm for Vertex Cover in the complement graph turns out to be a successful approach in practice for many such problems [ACF+04, AFLS07, ALSS06, CLS+05].

However, in many practical applications such as social [SF78] and biological [CLS+05] network analysis, cliques have been criticized for their overly restrictive nature or modeling disadvantages. Hence, more relaxed concepts of dense subgraphs such as s-plexes [SF78] are of interest, where one demands that each s-plex vertex does not need to be connected to all other vertices in the s-plex but to all but $s-1$. Thus, cliques are 1-plexes. One important part of this thesis follows this line of research; we theoretically analyze a generalized Vertex Cover problem, which asks for a minimum-cardinality vertex subset whose deletion results in a graph of bounded degree $s-1$; the complement of the resulting graph is an s-plex. We also implement an algorithm to solve this generalized Vertex Cover problem and show that one can use it to efficiently find large s-plexes in real-world graph instances. The generalized Vertex Cover problem can be formulated as a vertex deletion problem:

Π-Vertex Deletion
Instance: An undirected graph and a desired graph property Π.
Task: Delete a minimum number of vertices such that the remaining graph has property Π.

In other words, the task is to find a minimum number of vertices that "cover" all structures that violate Π. Vertex Cover is simply Π-Vertex Deletion where Π is the property "graph with no edges", and the problem that is used to find s-plexes as described above is Π-Vertex Deletion where Π is the property "graph of bounded degree $s-1$". Many more NP-hard graph problems can be expressed as such a vertex deletion problem, as for instance Vertex Bipartization and Undirected Feedback Vertex Set, where the corresponding property Π is "bipartite graph" and "cycle-free graph", respectively. Vertex deletion problems are one type of problems considered in this thesis. The other problems can be characterized as variants of the Maximum Matching problem.

Matching Problems. The Maximum Matching problem is a classical combinatorial problem in computer science with many applications (for instance, in computer vision [CWC+96], medicine [SGW+05], and computational biology [WMFH04]) and many theoretical results [LP86].

Maximum Matching
Instance: An undirected graph.
Task: Find a maximum number of vertex-disjoint edges in the graph.

Maximum Matching is polynomial-time solvable (e.g., [Edm65]). It can be considered as the "dual" of Vertex Cover with respect to linear programming covering-packing duality [Vaz01]. In this thesis, we consider mainly two types of NP-hard generalizations of Maximum Matching: instead of finding a maximum number of edges, one asks for

1. a maximum number of vertex-disjoint copies of some fixed graph occurring as subgraphs in the given graph, or

2. a maximum number of edges such that each pair of edges fulfills some distance constraints.

Such generalized matching problems have various applications in computational biology [ABWB+09, CR02], communication networks [BBK+04, SMS06, SSM06, KMPS04, KMPS07], and applications ranging from information theory to the design of efficient statistical experiments [Yus07].

NP-Hard Problems. All problems described above except Maximum Matching are NP-hard. It is a widely believed assumption that NP-hardness implies a combinatorial explosion in the solution space that leads to running times growing exponentially with the input size. Therefore, large instances of NP-hard problems cannot always be solved to optimality in a reasonable amount of time. In contrast, polynomial-time solvable problems like Maximum Matching are considered to be solvable efficiently. However, as the above generalizations of Vertex Cover and Maximum Matching, many problems of high practical relevance are NP-hard [GJ79]. There are several approaches to solve NP-hard problems in practice. The most common concepts for that purpose are

- heuristics,

- approximation algorithms, and

- fixed-parameter algorithms.

Heuristics drop the demand for good running time guarantees or useful quality guarantees. They are usually tuned to give good results on typical instances, but they might fail for some others. A simple heuristic for Vertex Cover works as follows. Start with an empty vertex set S, and then iteratively select a vertex v of maximum degree, add v to S and delete v and its incident edges from the graph, until the resulting graph contains no edges. The resulting set S is a vertex cover, that is, S covers all edges of the graph; however, one can construct instances for which this algorithm produces solutions that are far from optimal [PS98]. In general, heuristics have the drawback that there is no guarantee for good running times and/or good solution quality.

Approximation algorithms (see [ACG+99, Vaz01]) demand for good running time guarantees (that is, polynomial-time solvability), but instead cease the demand for an optimal solution, while still providing provable bounds on the solution quality. A simple approximation algorithm for Vertex Cover works as follows. Starting with an empty set S, while there exists an edge e in the graph, add e to S and remove e and both its endpoints from the graph, until there is are no more edges left. It is easy to see that S is a vertex cover. Each edge that has been added to S must be covered by at least one of its endpoints, but S contains *both* endpoints. Hence, S is at most twice as large as an optimal vertex cover. Thus, for *any* graph the algorithm finds a vertex cover that does not contain more than twice the number of vertices of an optimal vertex cover, hence this algorithm guarantees an approximation factor of two.

There are approximation algorithms that can find a solution with an arbitrarily good approximation factor in polynomial time (such an algorithm is called *polynomial-time approximation scheme*). Unfortunately, most of the problems considered in this paper do not admit such an approximation algorithm. For instance, the best-known approximation factor for Vertex Cover is two, and a lower bound of the factor is 1.36 assuming P $\neq$ NP [DS05], and these results translate to a large class of vertex deletion problems. Likewise, the generalized matching problems sketched above are generally APX-hard and do therefore not admit an arbitrarily good approximation algorithm. However, many applications of these problems usually demand for optimal or at least nearly optimal solutions.

Fixed-parameter algorithms [DF99, Nie06, FG06] find optimal solutions for NP-hard problems within provable running time bounds. As to be expected, such algorithms have exponential running times; however, for many problems (as most of the problems considered in this thesis) it is possible to confine the exponential running time part to a parameter, that is, the running time of the algorithm is exponential in the parameter, but only polynomial in the input size. If the parameter is small, which is often a reasonable assumption, then the algorithm runs efficiently, even for larger problem instances. For instance, for the Vertex Cover problem, a parameter k is the size of an optimal vertex cover. A very simple parameterized algorithm for Vertex Cover based on recursive branching works as follows. Select any edge e in the graph and branch into the two cases of putting one of the two endpoints of e into the vertex cover and delete the chosen vertex and all its incident edges. If there are no more edges to branch on while we deleted at most k vertices, then we have found a solution. A branching can be done in $O(n)$ time, and we obtain the overall running time $O(2^k \cdot n)$. For example, with this algorithm we can solve quite large instances with $k = 30$ and $n = 1000$ [Hüf07]. The best-known fixed-parameter algorithm for Vertex Cover runs in $O(1.274^k + kn)$ time [CKX06], which even allows to solve instance for $k = 120$ and $n = 1000$ [Hüf07].

Outline. In this thesis, we concentrate on the parameterized approach for the above-mentioned vertex deletion and matching problems. In the next chapter, we describe the parameterized approach in more detail and give a brief introduction to the most important techniques. We also provide the notation that is used throughout this thesis. Then, in Chapter 3, we give a short overview on the most important known results for vertex deletion and matching problems in the context of parameterized algorithmics, and a short description of our contributions. Chapters 4–6 mainly deal with vertex deletion problems, and Chapters 7 and 8 are mainly devoted to generalized matching problems. After that, we turn our attention to the experimental results presented in Chapter 9, where we describe the relation of Π-Vertex Deletion with Π being the property "graph of bounded degree $s-1$" and the problem of finding s-plexes and report about our experimental results. The thesis finishes in Chapter 10 with a brief outlook.

Chapter 2

Preliminaries

2.1 Basic Graph Theory

A graph is a tuple (V, E), where V is a finite set of *vertices* and E is a set of *edges*, which are size-two subsets of V, that is $E \subseteq \{\{u, v\} \mid \{u, v\} \subseteq V\}$. Note that by this definition the edges are not directed and there are no multiple edges or self-loops, that is, the graphs considered in this thesis are simple and undirected. The *complement* of a graph $G = (V, E)$ is the graph $\bar{G} := (V, \bar{E})$, where $\bar{E} := \{\{u, v\} \mid \{u, v\} \subseteq V\} \setminus E$. For a graph $G = (V, E)$, we write $V(G)$ to denote its vertex set and $E(G)$ to denote its edge set. By default, we use n and m to denote the number of vertices and edges, respectively, of a given graph. Two vertices $u, v \in V$ are *adjacent* if $\{u, v\} \in E$. A vertex $v \in V$ and an edge $e \in E$ are *incident* if $v \in e$.

For a vertex $v \in V(G)$, the set $N_G(v) := \{u \in V \mid \{u, v\} \in E\}$ is the set of *neighbors* of v. The *closed* neighborhood of v is defined as $N_G[v] := N_G(v) \cup \{v\}$. For $S \subseteq V$, the set $N_G(S) := \bigcup_{v \in S} N(v) \setminus S$ is the *neighborhood* of S. The *closed* neighborhood is denoted as $N_G[S] := N_G(S) \cup S$. If the graph G is clear from the context, then we also write $N(v)$, $N[v]$, $N(S)$, and $N[S]$ instead of $N_G(v)$, $N_G[v]$, $N_G(S)$, and $N_G[S]$, respectively. The *degree* of a vertex v is the number of its neighbors $|N(v)|$. If every vertex in G has degree at most d, then we say that G has *maximum degree* d. For a vertex set $S \subseteq V$, we write $G[S]$ to denote the graph induced by S in G, that is, $G[S] := (S, \{e \in E \mid e \subseteq S\})$. For a vertex $v \in V$, we also write $G - v$ instead of $G[V \setminus \{v\}]$ and for a vertex set $S \subseteq V$ we also write $G - S$ instead of $G[V \setminus S]$.

A *path* is a sequence of vertices $v_1, \ldots, v_p$ with $\{v_i, v_{i+1}\} \in E$ for all $1 \leq i < p$, where all the vertices v_i are distinct. The number of edges of a path is its *length*. A *cycle* is a path with $\{v_p, v_1\} \in E$. The *girth* of a graph is the length of a shortest cycle in it. A *clique* is a complete graph, that is, a graph in which all vertices are pairwise adjacent. A K_n is a clique of n vertices. The graph K_3 is also called *triangle*. A P_n is a path of n vertices, and C_n is

a cycle of n vertices. A *wheel* is a graph W that has a vertex $v \in V(W)$ that is adjacent to all other vertices such that $W - v$ is a cycle. For $s \geq 1$, the graph $K_{1,s} := (\{u, v_1, \ldots, v_s\}, \{\{u, v_1\}, \ldots, \{u, v_s\}\})$ is an *s-star*, or simply *star*. The vertex u is the *center* of the star and the vertices $v_1, \ldots, v_s$ are the *leaves* of the star. A *$\leq s$-star* is an s'-star with $s' \leq s$ and a *$<s$-star* is an s'-star with $s' < s$. A graph is *connected* if there is a path between any two vertices. For a connected graph G, a *cut-vertex* is a vertex $v \in V$ such that $G - v$ is not connected. The *distance* between two vertices u, v is the length of a shortest path between u and v. The *distance* between two edges e_1, e_2 is the smallest distance between any two vertices $u \in e_1$ and $v \in e_2$.

Given an undirected graph $G = (V, E)$ and an edge subset $E' \subseteq E$, to *subdivide* the edges E' in G means to remove from G all edges in E', and then to add for each edge $\{u, v\} \in E'$ a vertex $x_{u,v}$, making it adjacent to u and v. The vertices in $\{x_{u,v} \mid \{u, v\} \in E'\}$ are called *subdivision vertices*.

For a family of graphs $\mathcal{H}$ we define $V(\mathcal{H}) := \bigcup_{H \in \mathcal{H}} V(H)$ and $E(\mathcal{H}) := \bigcup_{H \in \mathcal{H}} E(H)$. We say that a graph H' is a *copy of H* if H' is isomorphic to H. For a graph G and a graph H, we say that H' is a *copy of H in G* if H' is a subgraph of G and H' is a copy of H. Given two graphs H_1 and H_2, the *intersection* of H_1 and H_2 is defined as $V(H_1) \cap V(H_2)$. A *packing* P of a graph H in a graph G is a set of pairwise vertex-disjoint copies of H in G.

Matching Basics. Given an undirected graph $G = (V, E)$, an edge subset $M \subseteq E$ is called a *matching* if the edges in M are pairwise disjoint. A matching M is *maximal* if there exists no edge $e \in (E \setminus M)$ such that $M \cup \{e\}$ is a matching. A matching M is *maximum* if there exists no larger matching. A vertex $v \in V$ is *matched* if there exists an edge in M that is incident to v. A vertex $v \in V$ is *unmatched* if it is not matched. An *M-alternating path* is a path in G that starts with an unmatched vertex, and then contains, alternately, edges from $E \setminus M$ and M. If an M-alternating path ends with an unmatched vertex, then it is called *M-augmenting path*.

Graph Properties. A *graph property* Π is a (possibly infinite) set of graphs. We also write that a graph G *satisfies* Π if $G \in \Pi$. A graph property Π is *hereditary* if it is closed under deleting vertices, that is, if $G \in \Pi$, then for any induced subgraph G' of G, $G' \in \Pi$. A hereditary graph property is *non-trivial* if it is satisfied by infinitely many graphs and it is not satisfied by infinitely many graphs. A hereditary graph property is *determined by the components* if a graph G satisfies Π whenever every connected component of G satisfies Π. For any hereditary property Π there exists a set of "minimal" forbidden induced subgraphs, that is, forbidden graphs for which every induced subgraph satisfies Π [GHK73]. If Π is a hereditary property that is determined by the components, then the corresponding set of forbidden induced subgraphs only contains *connected* graphs.

A graph is *planar* if it can be embedded in the plane, that is, it can be drawn

in a plane such that the edges only intersect in their endpoints. Every planar graph contains a vertex of degree at most five, which is a consequence of Euler's formula.

For more about graph theory, we refer to the books by Diestel [Die05] and West [Wes01].

2.2 Parameterized Complexity and Fixed-Parameter Algorithms

Since many graph problem are NP-hard, it seems hopeless to solve them exactly in polynomial time. However, NP-hardness expresses the computational hardness of a problem in the *worst case*, and there often exist even large instances of NP-hard problems that can be solved in reasonable time. The reason is that such instances might contain some structure that can be exploited by an algorithm. Such structure can often be expressed by a parameter (usually a nonnegative integer), and then one can do a two-dimensional worst-case analysis that measures the growth of the running time depending on the input size *and* the parameter. The hope is that the seemingly unavoidable combinatorial explosion of the running time can be restricted to the parameter. Then, if the parameter is small, which is often a reasonable assumption, the problem can be solved efficiently even on large instances.

For instance, for the Vertex Cover application sketched in Chapter 1 (sequence alignment), it is reasonable to assume that the solution is small; otherwise, one would have to remove too many sequences from the sample, which is an indication that the sample contains too many errors in order to derive meaningful results. Other types of parameters restrict the structure of the input graph; for instance, there are problems where the input typically has a tree-like structure which can be exploited to find an optimal solution (see also Section 2.3.4).

Downey and Fellows [DF99] first describe a formal framework for such a two-dimensional analysis of problems.

Definition 2.1. *A* parameterized problem *is a language $L \subseteq \Sigma^* \times \Sigma^*$, where Σ is a finite alphabet. The second component is called the* parameter *of the problem.*

Throughout this thesis the parameter is a nonnegative integer, and therefore we assume that $L \subseteq \Sigma^* \times \mathbb{N}$. For $(I, k) \in L$, the two dimensions of the parameterized complexity analysis are then the input size $n := |(I, k)|$ and the parameter k. Since in our applications all parameter values are upper-bounded by $|I|$, we can simply assume $n := |I|$ in our asymptotic considerations. The following notion expresses that a parameterized problem can be solved efficiently for small parameter values.

Definition 2.2. *A parameterized problem L is* fixed-parameter tractable *with respect to the parameter k if there exists an algorithm that decides in $f(k) \cdot \mathrm{poly}(n)$*

time whether $(I, k) \in L$, where f is a computable function only depending on k. The complexity class containing all fixed-parameter tractable problems is called FPT.

In other words, a parameterized problem is fixed-parameter tractable if it can be solved in time that is exponential in the parameter, but only polynomial in the input size. There are several techniques to show that a parameterized problem is fixed-parameter tractable. In the next section, we introduce some of the most important ones used in this thesis, like problem kernelization (Section 2.3.1), bounded search trees (Section 2.3.2), iterative compression (Section 2.3.3), and dynamic programming on tree decompositions (Section 2.3.4). There also exist parameterized problems that are likely to be not fixed-parameter tractable. Analogously to the concept of NP-hardness, Downey and Fellows [DF99] developed a framework containing reduction and completeness notions in order to show hardness of parameterized problems. See Section 2.3.5 for more details.

For a more detailed introduction to parameterized algorithmics and parameterized complexity theory we refer to the books by Downey and Fellows [DF99], Flum and Grohe [FG06], and Niedermeier [Nie06].

2.3 Basic Fixed-Parameter Techniques

In this section, we outline some of the most important techniques in the field of fixed-parameter algorithmics that are applied in this thesis. Concerning the first three techniques, see also a recent survey by Hüffner et al. [HNW08] for a more detailed description with many examples.

2.3.1 Problem Kernelization

To solve NP-hard problems, polynomial-time preprocessing is a natural approach. The main idea is to use preprocessing to remove the "easy" parts of the input in order to obtain the computationally hard "core" of the instance. One important requirement of such preprocessing in our context is that they preserve the ability to solve the problem to optimality, that is, that an optimal solution for the reduced instance can be used to derive an optimal solution for the input instance.

In the classic one-dimensional analysis of algorithms, it is difficult to measure the quality of such an "exact" polynomial-time preprocessing, since any preprocessing step with provable effectiveness (that is, a guarantee that the preprocessing step will reduce the instance) could be applied repeatedly until the remaining instance is empty, which would imply P = NP. The picture changes completely if we consider parameterized problems. Here, the parameter can be used to show provable size bounds of the instance after applying the preprocessing algorithm. Such a reduced instance is called *problem kernel*.

Definition 2.3. *Let $L \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem. A* reduction to a problem kernel *or* kernelization *is a polynomial-time transformation of an instance (I, k) to an instance (I', k') such that $(I, k) \in L$ if any only if $(I', k') \in L$, $|I'| \leq g(k)$ for some arbitrary computable function g depending only on k, and $k' \leq k$.*

Thus, a problem kernelization is an algorithm that replaces the input instance by an equivalent instance whose size depends only on k and not on the input size anymore. The *size* of the problem kernel is $|I'|$. However, for many graph problems, the kernel size is often stated with respect to the number of vertices only. Moreover, a problem kernel with $O(k)$ vertices is often called "linear problem kernel", although it might contain $O(k^2)$ edges. In this thesis, most of the kernelization results are stated with respect to the number of vertices.

A problem kernelization is often described via data reduction rules. A *data reduction rule* is an algorithm that replaces in polynomial time an instance (I, k) with an instance (I', k'), where $|I'| < |I|$, such that $(I, k) \in L$ if and only if $(I', k') \in L$. A problem instance to which none of a given set of reduction rules applies is called *reduced* with respect to these rules.

For an example of a problem kernel, consider the parameterized version of Vertex Cover, where the size of a vertex cover is bounded by the parameter k. If there is a vertex v of degree at least $k + 1$, then one may assume that v is in the vertex cover, since otherwise all neighbors of v have to be in the cover, which would be more than k. Therefore, as a first data reduction rule, we add all vertices with at least $k+1$ neighbors to the vertex cover, and for each vertex that is added, we decrease the parameter k by one. After that, a second data reduction rule deletes all degree-0 vertices, which is obviously correct. If the remaining graph is a yes-instance, that is, there exists a vertex cover S of size at most k, then the remaining graph contains at most k^2 edges and at most $k + k^2$ vertices, since each vertex in S has a most k neighbors, and there are no edges between vertices in $N(S)$. Therefore, a last data reduction rule returns the reduced graph if it contains at most k^2 edges and at most $k+k^2$ vertices; otherwise, it returns a trivial no-instance. The resulting instance is a $O(k^2)$-size problem kernel for Vertex Cover with respect to the parameter k. The currently best problem kernel for Vertex Cover has at most $2k$ vertices [NT75, CKJ01]. This kernelization has found practical applications in computational biology, where it helps to make problem instances small enough such that they can be solved exactly [AFLS07].

It is not difficult to see that any parameterized problem that admits a problem kernel is fixed-parameter tractable. The corresponding fixed-parameter algorithm simply solves the problem by brute force on the problem kernel. The contrary is also true:

Theorem 2.1 ([CCDF97])**.** *For every parameterized problem that is fixed-parameter tractable there exists a problem kernel and vice versa.*

Unfortunately, the theorem cannot be used to get an efficient fixed-parameter

algorithm from a problem kernel or a small (e.g., polynomial size) problem kernel from a fixed-parameter algorithm. It is mainly used to establish fixed-parameter tractability or the existence of a problem kernel.

Problem kernelization is a very powerful tool to show the effectiveness of data reduction rules. Moreover, since it preserves the ability to solve the problem exactly, virtually any method can be used to solve the problem on the kernel (like fixed-parameter algorithms, but also approximation and heuristic algorithms). However, problem kernelization is not restricted to serve as a pure preprocessing step. There is theoretical [NR00] and practical [ALSS06] evidence that it can be efficiently interleaved with the main solving algorithm (in particular bounded search trees, see Section 2.3.2). For instance, our experimental results in Chapter 9 are heavily based on such methods, achieving speedups of several orders of magnitude in practice.

A "success story" for kernelization is Cluster Editing, the problem of adding and deleting at most k edges of a graph such that every connected component becomes a clique. Here, a first problem kernel had $O(k^2)$ vertices [GGHN05], where k is the number of allowed editing operations. The kernelization has been gradually improved [FLRS07, PdSS09], and the best-known kernel size is now $4k$ vertices [Guo09]. Kernelization algorithms for Cluster Editing have also found applications in practice [BBK09, DLL+06]. As another example, for the (undirected) Feedback Vertex Set problem, a kernel of $O(k^3)$ vertices [Bod07] has recently been improved to $O(k^2)$ vertices [Tho09], where k is the feedback vertex set number of the given graph.

For more about kernelization refer to a survey by Guo and Niedermeier [GN07a].

2.3.2 Bounded Search Trees

It is usually inevitable to use some exponential-time method in order to solve an NP-hard problem to optimality. A standard way to do so is a systematic exhaustive search, which can be organized in a tree-like fashion. The basic idea is to find in polynomial-time a small part of the input such that at least one element of that part has to be in an optimal solution. We then branch into several cases of choosing an element of the small part to be in the solution, and then proceed recursively until a solution is found. A search tree corresponds to the recursive calls of such an algorithm. If we can bound the number of cases in each search tree node as well as the height of the tree (the maximum number of nested recursive calls), then we obtain a fixed-parameter algorithm. The number of recursive calls is the number of nodes in the according tree. This number is governed by linear recurrences with constant coefficients. These can be solved by standard mathematical methods [Nie06]. If the algorithm solves a problem instance of size s and calls itself recursively for problem instances of sizes $s - d_1, \ldots, s - d_i$, then $(d_1, \ldots, d_i)$ is called the *branching vector* of this recursion. It corresponds to the recurrence $T_s = T_{s-d_1} + \cdots + T_{s-d_i}$ for the asymptotic size T_s of the overall

search tree.

For instance, for parameterized Vertex Cover, where the size of a vertex cover is bounded by a parameter k, a small part of the input that contains at least one element of an optimal solution is a single edge in the graph, because we know that each edge must be covered by at least one of its endpoints. Thus, we select an arbitrary edge $e = \{u, v\}$ of the graph, and branch into the two subcases of adding u or v to the vertex cover. At least one of the two assumptions is correct. Then, if we decided that for example u is in the vertex cover, then we can delete u and all incident edges from the graph and proceed with the remaining instance. In each branching step, one vertex is deleted from the graph. After at most k recursive steps we either obtain an instance without edges, and we have found a solution, or the remaining instance has still some edges, which means that the corresponding path from the search tree root to the leaf cannot lead to a solution of size at most k. The branching vector corresponding to this recursion is $(1, 1)$. This leads to a search tree of size $O(2^k)$. The selection of an edge and the branching can be done in $O(n)$ time and therefore we obtain a fixed-parameter algorithm with running time $O(2^k \cdot n)$.

Bounded search tree algorithms for Vertex Cover combined with data reduction rules can be very fast in practice, allowing to solve real-world instances with several thousand vertices [ALSS06]. Our experimental results in Chapter 9 are also based on a bounded search tree approach combined with data reduction rules.

Again, a "success story" is Cluster Editing, the problem of adding and deleting at most k edges of a graph such that every connected component becomes a clique. A first search tree algorithm by Gramm et al. [GGHN05] runs in $O(2.27^k + n^3)$ time [DLL+06], which was improved to $O(1.92^k + n^3)$ by a computer-generated search tree algorithm [GGHN04]. The approach by Gramm et al. [GGHN05] has been adapted to weighted instances [RWB+07]. Böcker et al. [BBBT09] improved the running time of the weighted version to $O(1.82^k + n^3)$. This approach has also been experimentally tested and compared with other approaches [BBK09].

2.3.3 Iterative Compression

Until 2004, the parameterized complexity of several important NP-hard minimization problems was open. Then, Reed, Smith, and Vetta [RSV04] introduced in a very short paper a new technique that is now called iterative compression. Meanwhile, based on this technique, a number of the mentioned open questions could be positively answered by giving corresponding fixed-parameter algorithms. To become more specific, let us consider the NP-complete Vertex Bipartization problem.

Vertex Bipartization
Input: An undirected graph $G = (V, E)$ and an integer $k \geq 0$.
Question: Does there exist a vertex subset $S \subseteq V$ of size at most k such that $G - S$ is bipartite?

The central part of iterative compression is a *compression routine*, which, given a graph and a size-$(k+1)$ solution of Vertex Bipartization, either computes a smaller solution or proves that no smaller solution exists. This compression routine runs in $O(3^k \cdot km)$ time [RSV04]. An algorithm to solve Vertex Bipartization uses the compression routine as follows. Start with $V' = \emptyset$ and $S = \emptyset$; clearly, the empty set is a solution for an empty graph. Iterating over all graph vertices, step by step add one vertex $v \notin V'$ from V to both V' and S. Then, S is still a solution for $G[V']$, although possibly not a minimum one. If in a step $|S| \leq k$, then proceed with the next step; otherwise, use the compression routine with input $G[V']$ and S to obtain a smaller solution, if it exists. If there exists a smaller solution than $k+1$, then proceed with the next step using the smaller solution; otherwise, $G[V']$ is a no-instance and therefore G is also a no-instance. Since eventually $V' = V$, one obtains a solution for G once the algorithm returns S. The algorithm runs in $O(3^k \cdot knm)$ time. With an improved analysis, this can be improved to $O(3^k \cdot nm)$ time [RSV04, Hüf09]. In other words, Vertex Bipartization is fixed-parameter tractable with respect to the parameter k. Note that there exist several variants of the original proof [RSV04] that strive for an improved presentation [Hüf09, LSS09].

Similar breakthroughs were achieved, e.g., for the NP-complete problems Undirected Feedback Vertex Set [DFL+07, GGH+06, CFL+08], Directed Feedback Vertex Set [CLL+08], and Almost 2-Sat [RO09]. See also [GMN09] for a survey about iterative compression.

While applying the compression routine is usually straightforward, finding a compression routine is not. It is not even clear that a compression routine with useful running times exists even if we know that the corresponding parameterized problem is fixed-parameter tractable. In Chapter 6, we give an overview about known results that have been achieved using iterative compression, and we give an iterative compression framework for vertex deletion problems. Moreover, for a large class of vertex deletion problems that are covered by the framework, we analyze the computational complexity of the compression task, which is the computational task that is solved by the compression routine.

2.3.4 Tree Decomposition Based Algorithms

Many NP-hard graph problems such as Vertex Cover can be solved in polynomial time on trees. This motivates to study such problems in graphs that are similar to trees. Tree decompositions are a formal way to describe the "tree-likeness" of a given graph [Bod06, Klo94].

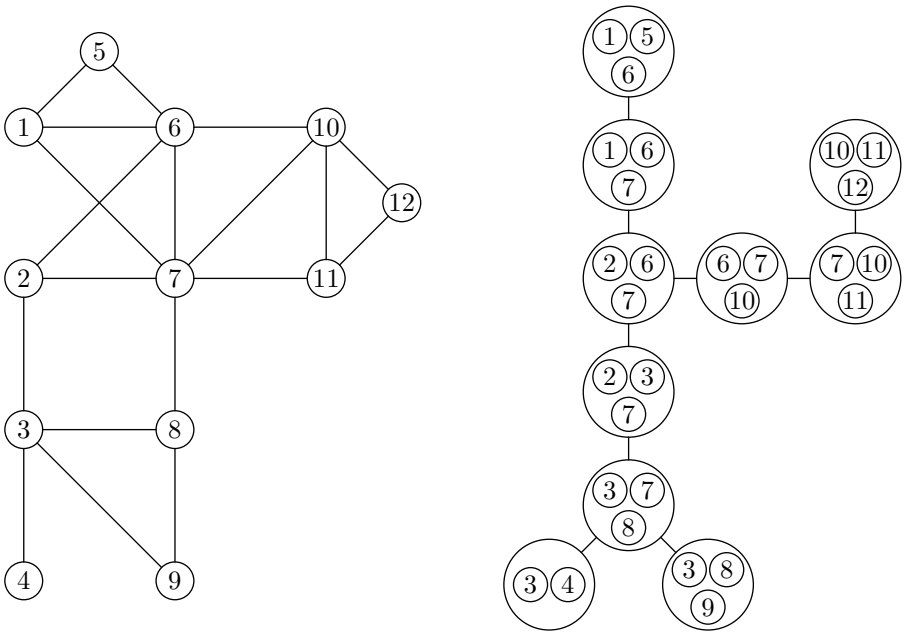


Figure 2.1: Example of a graph (left) and a corresponding tree decomposition of width two (right).

Definition 2.4. *Let $G = (V, E)$ be a graph. A* tree decomposition *of G is a pair $(\{X_i \mid i \in I\}, T)$, where each X_i is a subset of V, called a* bag*, and T is a tree with the elements of I as* nodes*. The following three properties must hold:*

1. $\bigcup_{i \in I} X_i = V$,
2. *for every edge $e \in E$ there is an $i \in I$ such that $e \subseteq X_i$, and*
3. *for all $i, j, k \in I$, if j lies on the path from i to k in T, then $X_i \cap X_k \subseteq X_j$.*

The width *of $(\{X_i \mid i \in I\}, T)$ equals $\max\{|X_i| \mid i \in I\} - 1$. The* treewidth *of G is the minimum k such that G has a tree decomposition of width k.*

Note that trees have treewidth one. In Figure 2.1 we give an example of a graph and a tree decomposition of width two. An important property of tree decompositions of width k is that each bag X_i that is not a leaf of the decomposition tree is a separator containing at most $k + 1$ vertices, that is, after the deletion of the vertices in X_i the graph decomposes into at least two connected components (for example, the bag $\{2, 6, 7\}$ in Figure 2.1 is clearly a separator). As one consequence, in order to solve some problem on a graph, we process the bags in the tree decomposition in a bottom-up manner and do dynamic programming, storing all possible solutions for each bag X_i in a table that uses $f(k)$ space for some function f only depending on k. This yields a fixed-parameter algorithm with respect to the parameter k. For example, for Vertex Cover one can use a table of 2^{k+1} entries for each bag X_i, one for each possible intersection of X_i with

a vertex cover, and an entry contains the minimum size of a vertex cover in the graph corresponding to the subtree rooted at i in the decomposition tree T. Once we computed all entries for a node i, one can reuse them to compute solutions on X_j, where j is the parent node of i.

A limiting factor of the dynamic programming technique using tree decompositions is the construction of tree decompositions of *small* width. Given an undirected graph G and an integer k, the problem to determine whether the treewidth of G is at most k is NP-complete [ACP87]. There exist various algorithms (e.g., heuristics or approximation algorithms) to compute a tree decomposition of small width. See, e.g., the survey by Bodlaender and Koster [BK08]. The problem to determine whether the treewidth of a graph is at most k is fixed-parameter tractable with respect to the treewidth k as the parameter [Bod96], and a tree decomposition can be constructed in $f(k) \cdot (n+m)$ time, where f is a function only depending on k.

For the description of tree decomposition based algorithms it is convenient to use a *nice tree decomposition*, which has a particularly simple structure [Klo94].

Definition 2.5. *A tree decomposition* $(\{X_i \mid i \in I\}, T)$ *is called a* nice tree decomposition *if the following conditions are satisfied (we suppose the decomposition tree* T *to be rooted at some arbitrary but fixed node):*

1. *Every node of the tree* T *has at most two children.*
2. *If a node* i *has two children* j *and* k*, then* $X_i = X_j = X_k$ *(in this case* i *is called a* join node*).*
3. *If a node* i *has one child* j*, then either*
   (a) $|X_i| = |X_j| + 1$ *and* $X_j \subset X_i$ *(in this case* i *is called an* introduce node*), or*
   (b) $|X_i| = |X_j| - 1$ *and* $X_i \subset X_j$ *(in this case* i *is called a* forget node*).*

A part of a nice tree decomposition for the example in Figure 2.1 is given in Figure 2.2. A given tree decomposition can be transformed into a nice tree decomposition in linear time:

Lemma 2.1 (Lemma 13.1.3 of [Klo94])**.** *Given a tree decomposition of a graph* G *that has width* ω *and* $O(n)$ *nodes, where* n *is the number of vertices of* G*. Then we can find in* $O(n)$ *time a nice tree decomposition of* G *that also has width* ω *and* $O(n)$ *nodes.*

2.3.5 Parameterized Reductions and W-Hardness

In analogy to the concept of NP-hardness in classical "one-dimensional" complexity theory, Downey and Fellows [DF99] developed a framework providing a reducibility and completeness program.

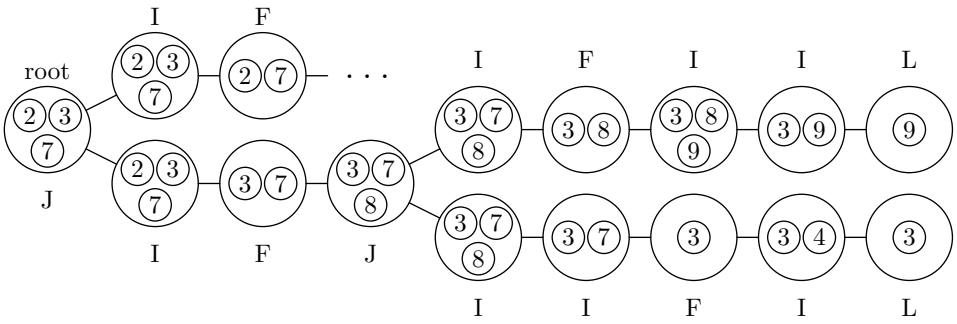


Figure 2.2: A part of a nice tree decomposition of the graph in Figure 2.1. Each node is marked with a letter denoting the node type: leaf node (L), introduce node (I), forget node (F), and join node (J).

Definition 2.6. *A* parameterized reduction *from a parameterized problem* $L \subseteq \Sigma^* \times \mathbb{N}$ *to another parameterized problem* $L' \subseteq \Sigma^* \times \mathbb{N}$ *is a function that, given an instance* (I, k)*, returns in time* $f(k) \cdot \text{poly}(|(I, k)|)$ *an instance* (I', k') *(with* k' *depending only on* k*) such that* $(I, k) \in L$ *if and only if* $(I', k') \in L'$*.*

A parameterized problem L belongs to $W[t]$ if L reduces to a weighted satisfiability problem for the family of circuits of weft at most t and depth at most some function of the parameter k. For more about this characterization, see the book by Downey and Fellows [DF98]. For this thesis, it is only important to know that W[t]-hard problems are unlikely to be fixed-parameter tractable, for $t \geq 1$.

In this thesis, we reduce from two standard parameterized problems to show W[t]-hardness for $t \in \{1, 2\}$.

Independent Set
Input: An undirected graph $G = (V, E)$ and an integer $k \geq 0$.
Question: Does there exist a vertex subset $S \subseteq V$ of size at least k such that there are no edges in $G[S]$?

Dominating Set
Input: A graph $G = (V, E)$ and an integer $k \geq 0$.
Question: Does there exist a vertex subset $S \subseteq V$ of size at most k such that every vertex of V either belongs to S or has a neighbor in S?

The Independent Set problem is W[1]-complete and the Dominating Set problem is W[2]-complete [DF98]. For more about parameterized complexity theory, also refer to the book by Flum and Grohe [FG06] and a survey by Chen and Meng [CM08].

Chapter 3

Overview

In this chapter, we give a short survey of general results concerning vertex deletion (Section 3.1) and packing problems (Section 3.2), with a focus on results in the context of parameterized complexity analysis. Section 3.3 provides a short summary of the results presented in this thesis.

3.1 Vertex Deletion Problems

The decision version of Π-Vertex Deletion (cf. Chapter 1) can be formulated as follows. Let Π be any graph property.

> Π-Vertex Deletion
> **Input:** An undirected graph $G = (V, E)$ and an integer $k \geq 0$.
> **Question:** Does there exist a vertex subset $S \subseteq V$ of size at most k such that $G - S \in \Pi$?

Lewis and Yannakakis [LY80] showed that if Π is a non-trivial hereditary graph property that can be tested in polynomial time, then Π-Vertex Deletion is NP-complete. Analogous results exist for bipartite graphs, where only Vertex Cover is solvable in polynomial time; all other Π-Vertex Deletion problems are NP-complete [Yan81b]. Concerning approximation, it follows from the reductions by Lewis and Yannakakis [LY80], which are approximation-preserving, that no vertex deletion problem on hereditary properties can be approximated better than Vertex Cover (best-known approximation factor 2). For the same reason, the APX-hardness of Vertex Cover [PY91] and the polynomial-time approximation lower bound of 1.36 assuming P $\neq$ NP [DS05] carry over to vertex deletion problems on nontrivial hereditary properties (see also [LY93]). A generic approximation framework has been presented by Fujito [Fuj98], which yields non-trivial constant-factor approximation algorithms for various properties. Moreover, there exist infinitely many vertex deletion problems on nontrivial hereditary properties with infinitely many minimal forbidden induced subgraphs that are factor-2 approximable within polynomial time [Fuj99].

If the set of forbidden induced subgraphs corresponding to the property Π is finite, then Π-Vertex Deletion is fixed-parameter tractable with respect to the parameter k: in this case, the forbidden induced subgraphs have constant size, and one can use a simple bounded search tree algorithm that finds a forbidden induced subgraph, and branches into all cases of deleting a vertex of this subgraph. Cai [Cai96] showed that this also works if the set of forbidden induced subgraphs is not known (one has only an algorithm that can test in polynomial time whether a graph is in Π).

If the property Π is closed under taking minors (that is, closed under deleting vertices, deleting edges, and contracting edges), then due to a deep result by Robertson and Seymour [RS04] there exists an algorithm that decides Π-Vertex Deletion in time $f(k) \cdot n^3$; hence, Π-Vertex Deletion is fixed-parameter tractable with respect to the parameter k. However, this result is non-constructive and the hidden constants are huge. An example for a minor-closed property Π is "planar"; for this property there also exists a constructive proof by showing an fixed-parameter algorithm for Π-Vertex Deletion that runs in in $f(k) \cdot n^2$ time [MS07a].

There exist more results for other particular properties Π. For example, Undirected Feedback Vertex Set corresponds to the case that Π means "cycle-free" whereas for Vertex Bipartization Π means "bipartite". Undirected Feedback Vertex Set has been intensively studied in terms of parameterized algorithms [Bod94, BBYG00, RSS02, KPS04, RSS05, DFL$^+$07, GGH$^+$06, CFL$^+$08] and kernelization [BECF$^+$06, Bod07, Tho09]. For Vertex Bipartization there exist several variants of a fixed-parameter algorithm [RSV04, Hüf09, LSS09], which has also been implemented and tested successfully on real-world instances [Hüf07, Hüf09]. These algorithms are based on the iterative compression technique. See Chapter 6 for more details.

We refer to Table 3.1 for an overview on known and new parameterized complexity results for particular vertex deletion problems.

Related Problems. There also exists a classification of the parameterized complexity for the dual parameterization of Π-Vertex Deletion: the input is an undirected graph G and an integer $k \geq 0$, and one asks whether there exists a size-k induced subgraph of G that fulfills Π. For instance, if Π is the set of all edge-free graphs (that is, the property "edge-free"), then the problem is the dual parameterization of Vertex Cover, that is, the W[1]-complete Independent Set problem. Khot and Raman [KR02] showed that the problem is W[1]-complete if Π contains all edge-free graphs but not all complete graphs, and fixed-parameter tractable if Π contains all complete graphs but not all edge-free graphs.

A class of problems that is closely related to vertex deletion problems are graph modification problems. Here, for some fixed graph property Π, the input is a graph, and the task is to apply a minimum number of modifications to the graph such that Π is fulfilled. Allowed graph modifications are usually

Table 3.1: Fixed-parameter algorithms and problem kernel sizes for various vertex deletion problems. Bounded-Degree-d Vertex Deletion is abbreviated by BDD-d, Regular-Degree-d Vertex Deletion by RDD-d, where d is a constant. For the exact definitions of the problems considered in this thesis, refer to the corresponding chapter.

problem	property Π	algorithm	kernel (vertices)
Vertex Cover	edge-free	$O(1.2738 + kn)$ [CKX06]	$2k$ [CKJ01]
Vertex Bipartization	bipartite	$O(3^k \cdot mn)$ [RSV04, Hüf09, LSS09]	open
Feedback Vertex Set	cycle-free	$O(5^k \cdot kn^2)$ [CFL$^+$08]	$O(k^2)$ [Tho09]
Chordal Deletion	chordal	$f(k) \cdot \text{poly}(n)$ [Mar09]	open
Planar Deletion	planar	$f(k) \cdot n^2$ [MS07a]	open
Wheel-Free Deletion	wheel-free	W[2]-hard [Lok08]	
Cluster Vertex Deletion	(induced P_3)-free	$O(2^k \cdot k^6 \log k + nm)$ [Ch. 6]	$O(k^2)$ [HKMN09a]
BDD-1	(non-induced P_3)-free	$O(2^k \cdot k^2 + kn)$ [Ch. 4]	$15k$ [Ch. 4]
BDD-d	(non-induced $K_{1,(d+1)}$)-free	Hitting Set [Ch. 4]	$O(k^{1+\epsilon})$ [Ch. 4]
RDD-d	d-regular	$O((d+2)^k \cdot kn)$ [Ch. 5]	$O(k^3)$ [Ch. 5]
$\mathcal{H}$-Free Vertex Deletion	$\mathcal{H}$-free	Hitting Set [Ch. 7]	$O(k^{h-1})$ [Ch. 7]

edge insertion, edge deletion, and vertex deletion. In this thesis, we restrict our attention to graph modification problems restricted to vertex deletion as the only allowed modification. Graph modification problems have been intensively studied in literature, see, e.g., [Sha02, Guo06].

Vertex deletion problems have also been studied in other classes of graphs, in particular, directed graphs. For instance, the parameterized complexity of Directed Feedback Vertex Set, the problem of delete a minimum number of vertices in order to remove all directed cycles from a directed graph, was a long-standing open problem. Chen et al. [CLL+08] settled this open problem by showing that Directed Feedback Vertex Set is fixed-parameter tractable with respect to the number of deleted vertices as parameter. The algorithm by Chen et al. [CLL+08] has also been implemented and tested in practice [FWY09]. Directed Feedback Vertex Set has also been studied in tournaments, that is, directed graphs in which there exists a directed edge between any two vertices. In such graphs, Directed Feedback Vertex Set can be solved in $O(2^k \cdot n^2(\log n + k))$ time, where k is the number of deleted vertices, and admits a problem kernel of $O(k^3)$ vertices [DGH+09].

3.2 Generalized Matching Problems

The first generalized matching problem that will be studied in this thesis, H-Packing, asks for at least k vertex-disjoint copies of a fixed graph H in a given graph G. If H is a connected graph with at least three vertices, then H-Packing becomes NP-complete [KH78] and also APX-complete [Kan94]. There exists a simple polynomial-time approximation algorithm that greedily finds an inclusion-maximal set of copies of H in G, yielding an approximation factor of $|V(H)|$ (cf. [Yus07]). There are further results for particular graphs H, e.g., if H is a clique. See Chapter 7 for more details.

Concerning parameterized complexity, the H-Packing problem can be solved in $2^{|V(H)|\cdot k} \cdot \text{poly}(n)$ time with a randomized algorithm [Kou08] and in $2^{2|V(H)|\cdot k} \cdot \text{poly}(n)$ time [CKL+09] with a deterministic algorithm, which has been improved to $2^{(2|V(H)|-1)\cdot k} \cdot \text{poly}(n)$ [FLLW09]. The deterministic algorithms also work for a weighted variant of the problem within the same running time bounds. Hence, H-Packing is fixed-parameter tractable with respect to parameter k. Further results exist for particular graphs H. See Chapter 7 for more details.

The second generalized matching problem considered in this thesis, Induced Matching, asks for at least k edges in a given graph such that the edges have pairwise distance at least two. This problem is NP-hard [SV82] and cannot be approximated to within a factor of $n^{1/2-\epsilon}$ for any $\epsilon > 0$ [OFGZ08] in general graphs. It has various applications, mainly in (wireless) networks. The problem has been studied intensively in restricted graph classes with respect to its "classical" computational complexity and approximation. In Chapter 8 we give a more detailed overview.

There are numerous related problems, for instance, packing of graphs from a given family $\mathcal{H}$ (each copy must be isomorphic to some $H \in \mathcal{H}$) [LP09], packing on directed graphs [BHR08], factor problems (every vertex must be part of some copy of H) [CTW09, GRC+98], edge-disjoint packing [MPS04], and many more (see [Yus07]). There exists also some research concerning problems that combine H-Packing with distance constraints, like Triangle Packing with the additional constraints that the triangles should have distance at least two [CH06, Cam09].

3.3 Summary of Results

An important part of this thesis is on problem kernelization for various vertex deletion and generalized matching problems.

The first part of this thesis is mainly devoted to vertex deletion problems for degree-based graph properties. The first studied problem is Bounded-Degree-d Vertex Deletion, where the task is to delete at most k vertices such that the remaining graph has maximum degree d. This problem is mainly motivated by an application in finding dense subgraphs, for instance in biological and social networks. The main result in Chapter 4 is a generalization of a local optimization algorithm for Vertex Cover theorem by Nemhauser and Trotter [NT75], which yields an almost linear problem kernel for Bounded-Degree-d Vertex Deletion, that is, a problem kernel of $O(k^{1+\epsilon})$ vertices for any constant $\epsilon > 0$. The problem can be solved with a fixed-parameter algorithm based on a simple bounded search tree that is also presented in that chapter; such an approach together with various data reduction rules is then implemented and tested on real-world instances in Chapter 9. An additional result is a fixed-parameter algorithm based on iterative compression for the special case $d = 1$, running in $O(2^k \cdot k^2 + kn)$ time. Moreover, it is shown that Bounded-Degree-d Vertex Deletion becomes W[2]-complete for unbounded d.

Chapter 5 is devoted to a vertex deletion problem for a non-hereditary degree-based graph property, namely Regular-Degree-d Vertex Deletion, where the task is to delete at most k vertices such that the remaining graph is regular with constant degree d. Since the graph property is not hereditary, the general NP-hardness framework by Lewis and Yannakakis [LY80] does not apply. Therefore, we provide an NP-hardness proof for Regular-Degree-d Vertex Deletion. This problem is similar to Bounded-Degree-d Vertex Deletion in some sense, but it has some very different properties due to the fact that the underlying graph property is non-hereditary. The main result in Chapter 5 is an $O(k^3)$-vertex problem for Regular-Degree-d Vertex Deletion for constant d. Furthermore, we show a fixed-parameter algorithm based on a bounded search tree approach.

Chapter 6 deals with the iterative compression technique. Since iterative compression has been particularly useful to derive fast fixed-parameter algorithms

for vertex deletion problems, we first give an overview on known applications of that technique. Then, we investigate the computational complexity of a general "compression task" centrally occurring in all known applications of iterative compression. The core issue (particularly but not only motivated by iterative compression) is to determine the computational complexity of, given an already inclusion-minimal solution for an underlying (typically NP-hard) vertex deletion problem in graphs, to find a better *disjoint* solution. The complexity of this task so far has been lacking a systematic study. We consider a large class of vertex deletion problems for hereditary graph properties on undirected graphs and show that, except for few cases which are polynomial-time solvable, the others are NP-complete. This class includes problems such as Vertex Cover (here the corresponding compression task is decidable in polynomial time) or Undirected Feedback Vertex Set (here the corresponding compression task is NP-complete).

Chapter 7 is devoted to H-Packing, the problem of packing at least k copies of a fixed graph H into a given graph. We first give a problem kernel of $O(k^2)$ vertices for Triangle Packing, that is, H-Packing with H being a triangle. This improves known results, and the advantage of our technique is that it can be applied to general H-Packing, yielding a kernel of $O(k^{|V(H)|-1})$ vertices. As a side result, we obtain a new type of kernel for Hitting Set, which can be used to kernelize a large class of vertex deletion problems.

Chapter 8 deals with the Induced Matching problem, the problem of packing at least k edges such that the edges have pairwise distance at least two. This problem is W[1]-hard with respect to the parameter k in general graphs. Therefore, we study the parameterized complexity of Induced Matching in restricted graph classes. We provide fixed-parameter tractability results for planar graphs, bounded-degree graphs, graphs with girth at least six, bipartite graphs, line graphs, and graphs of bounded treewidth. In particular, we give a linear-size problem kernel for planar graphs.

In Chapter 9 we apply methods of algorithm engineering for the problem of finding maximum s-plexes in a graph. An s-plex denotes a vertex subset in a graph inducing a subgraph where every vertex has edges to all but at most s vertices in the s-plex. Cliques are 1-plexes. In analogy to the special case of finding maximum-cardinality cliques, finding maximum-cardinality s-plexes is NP-hard. This problem can be formulated as the dual parameterization of Bounded-Degree Vertex Deletion. Our implementation is based on the theoretical work from Chapter 4, and experiments indicate the competitiveness of our approach, for many real-world graphs outperforming the previously used methods.

Chapter 4

Bounded-Degree Vertex Deletion

In this chapter, we consider the problem of deleting at most k vertices from a given graph in order to obtain a graph of maximum degree d for any fixed d. The underlying graph property "each vertex has degree at most d" is hereditary. We call this problem Bounded-Degree-d Vertex Deletion. Clearly, for $d = 0$, this problem is equivalent to Vertex Cover. For Vertex Cover, there exists a classical local optimization theorem by Nemhauser and Trotter [NT75], which, among other things, delivers a $2k$-vertex problem kernel. The main contribution in this chapter is a generalization of the theorem by Nemhauser and Trotter for Bounded-Degree-d Vertex Deletion. In terms of problem kernelization, our generalization yields a linear-vertex problem kernel for Bounded-Degree-d Vertex Deletion for $d \leq 1$, and an *almost* linear-vertex problem kernel for $d \geq 2$, that is, we can show that there exists a problem kernel of $O(k^{1+\epsilon})$ vertices for any constant $\epsilon > 0$. Moreover, we show various fixed-parameter algorithms for Bounded-Degree-d Vertex Deletion, with a focus on the case $d = 1$, where we obtain an algorithm running in $O(2^k \cdot k^2 + kn)$ time using iterative compression. On the negative side, we show that Bounded-Degree-d Vertex Deletion is W[2]-complete with respect to the parameter k if the maximum degree d is unbounded.

The generalization of the Nemhauser-Trotter theorem is the main result in this chapter. It is presented in Section 4.3. After that, two search tree algorithms are given in Section 4.4, followed by the algorithm based on iterative compression in Section 4.5. Finally, we present our completeness result in Section 4.6.

4.1 Introduction and Known Results

A *bdd-d-set* for a graph $G = (V, E)$ is a vertex subset whose removal from G yields a graph in which each vertex has degree at most d. The central problem of this chapter is defined as follows.

Bounded-Degree-d Vertex Deletion
Input: An undirected graph $G = (V, E)$ and an integer $k \geq 0$.
Question: Does there exist a bdd-d-set $S \subseteq V$ of size at most k for G?

Applications. To advocate and justify research on Bounded-Degree-d Vertex Deletion, we describe an application in computational biology (also see Chapter 9). In the analysis of genetic networks based on micro-array data, a clique-centric approach has shown great success [BCK+05, CLS+05]. Roughly speaking, finding cliques or near-cliques (called paracliques [CLS+05]) has been a central tool. Since finding cliques is computationally hard (also with respect to approximation), Chesler et al. [CLS+05, page 241] stated that "cliques are identified through a transformation to the complementary dual Vertex Cover problem and the use of highly parallel algorithms based on the notion of fixed-parameter tractability." More specifically, in these Vertex Cover-based algorithms polynomial-time data reduction (such as the NT-Theorem) plays a decisive role [Lan08] (also see [AFLS07]) for efficient solvability of the given real-world data. However, since biological and other real-world data typically contain errors, the demand for finding cliques (that is, fully connected subgraphs) often seems overly restrictive and somewhat relaxed notations of cliques are more appropriate. For instance, Chesler et al. [CLS+05] introduced paracliques, which are achieved by greedily extending the found cliques by vertices that are connected to almost all (para)clique vertices. An elegant mathematical concept of "relaxed cliques" is that of s-plexes [SF78], where one demands that each s-plex vertex does not need to be connected to all other vertices in the s-plex but to all but $s - 1$. Thus, cliques are 1-plexes. The corresponding problem to find maximum-cardinality s-plexes in a graph is basically as computationally hard as clique detection is [BBH09, CST+07, KHMN09]. However, as Vertex Cover is the complementary dual problem for clique detection, Bounded-Degree-d Vertex Deletion is the dual problem for s-plex detection: an n-vertex graph has an s-plex of size k if and only if its complement graph has a solution set for Bounded-Degree-d Vertex Deletion with $d = s - 1$ of size $n - k$, and the solution sets can directly be computed from each other. The Vertex Cover polynomial-time data reduction algorithm has played an important role in the practical success story of analyzing real-world genetic and other biological networks [BCK+05, CLS+05]. Our new polynomial-time data reduction algorithms for Bounded-Degree-d Vertex Deletion have the potential to play a similar role. We implemented most of the algorithms presented in this chapter and obtained promising results, see Chapter 9 for further details.

Hardness and Approximation. Bounded-Degree-d Vertex Deletion is a vertex deletion problem for the hereditary graph property "each vertex has degree at most d" and therefore NP-complete due to a general result by Lewis and Yannakakis [LY80]. Bounded-Degree-1 Vertex Deletion is the parame-

Table 4.1: Our parameterized complexity results for Bounded-Degree-d Vertex Deletion. Here, the parameter k denotes the size of a minimum bdd-d-set, and $\epsilon > 0$ is an arbitrary constant.

case	algorithm	problem kernel (vertices)
$d = 1$	$O(2^k \cdot k^2 + kn)$ [Sec. 4.5]	$15k$ [Sec. 4.3]
$d \geq 2$	$O((d+2)^k + n(k+d))$ [Sec. 4.4]	$O(k(k+d)d)$ [Sec. 4.2] $O(k^{1+\epsilon})$ [Sec. 4.3]
d unbounded	W[2]-complete [Sec. 4.6]	

terized dual of Dissociation Number and therefore NP-complete on C_4-free bipartite graphs of maximum degree three [Yan81a, BCL04].

Concerning approximation, it follows from the reductions by Lewis and Yannakakis [LY80], which are approximation-preserving, that Bounded-Degree-d Vertex Deletion cannot be approximated better than Vertex Cover (approximation lower bound of 1.36 assuming P $\neq$ NP [DS05]). For the same reason, the APX-hardness of Vertex Cover [PY91] is carried over to Bounded-Degree-d Vertex Deletion. For $d \leq 1$, the best-known approximation factor is 2 [Fuj98], and for $d \geq 2$ the best-known approximation factor is $2 + \log(d)$ [OB03].

Parameterized Complexity. Bounded-Degree-d Vertex Deletion is fixed-parameter tractable with respect to the parameter k for constant d, which can be seen easily by reduction to $(d+2)$-Hitting Set by enumerating all $(d+1)$-stars (a star with $d+1$ leaves) in the given graph; a $(d+1)$-star is the minimal forbidden subgraph for Bounded-Degree-d Vertex Deletion. The corresponding algorithm runs in $O(c^k + n^{d+2})$, where $c = d + 1 + O(1/(d+2))$ for $d \geq 2$ [NR03] and in $O(2.08^k + n^3)$ for $d = 1$ [Wah07]. There also exists a search tree algorithm with running time $O((d+k)^{k+1} \cdot n)$ [NRT05] and an algorithm that enumerates all minimal solutions in $O((d+2)^k \cdot (k+d)^2 \cdot m)$ time [KHMN09].

We show a simple $O(k^2)$-vertex problem kernel for Bounded-Degree-d Vertex Deletion in Section 4.2, because we use the corresponding reduction rules in our implementation (Chapter 9). Moreover, one can use the $O(k^2)$-vertex problem kernel to (quickly) kernelize a graph before applying other (slower) reduction rules (we apply this idea in Section 4.5). We provide search tree algorithms for general Bounded-Degree-d Vertex Deletion and for the case $d = 1$ in Section 4.4. These algorithms do not improve the exponential running time part compared to the simple reduction to Hitting Set mentioned above. However, for practical purposes as described in Chapter 9, these search tree algorithms are better suited, because they are simple to implement. See Table 4.1 for an overview on all results in this chapter.

4.2 A Quadratic-Vertex Problem Kernel

Bounded-Degree-d Vertex Deletion admits a very simple $O(k(k+d)d)$-vertex problem kernel, which is a generalization of the well-known quadratic-vertex kernel for Vertex Cover attributed to Buss [BG93]. The corresponding data reduction rules are as follows.

Reduction Rule 4.1. *If there exists a vertex $v \in V$ of degree more than $d+k$, then delete v from G and decrease k by one.*

This data reduction rule is correct, because v is always in an optimal bdd-d-set S: if one assumes that $v \notin S$, then at least $k+1$ of the neighbors of v would have to be contained in S, a contradiction. Therefore, v is contained in every minimum-cardinality bdd-d-set and can be safely deleted, decreasing the parameter k by one.

Reduction Rule 4.2. *If there exists a vertex $v \in V$ such that each vertex in $N[v]$ has maximum degree d, then delete v from G.*

This rule is obviously correct, as no minimum-cardinality solution would contain v, and the neighbors of v still have bounded degree d after the deletion of v.

Lemma 4.1. *Reduction Rule 4.1 and Reduction Rule 4.2 can be applied in $O(n(k+d))$ time.*

Proof. In order to show the running time, we claim that a yes-instance contains at most $n(k+d)$ edges. To this end, assume that G contains more than $n(k+d)$ edges. Then, G contains an induced subgraph of minimum degree $d+k+1$: to see this, simply delete all vertices of degree at most $d+k$ from G; this removes at most $n(d+k)$ edges, hence, after the deletion of all vertices of degree at most $d+k$, there has to remain an induced subgraph of G of minimum degree $d+k+1$. In this subgraph of minimum degree $d+k+1$, assuming that any vertex is not in a bdd-d-set S yields that more than k of its neighbors have to be in S. Thus, there is no bdd-d-set of size at most k in this graph. Thus, the graph G is a no-instance. This shows the claim.

Next, we show the running time. The first step is to test whether $E(G) > n(k+d)$, and if so, abort returning "no-instance". This is correct due to our claim and the test can be performed in $O(n(k+d))$ time (simply count the edges and abort if there are too many). Otherwise, proceed by iterating over all vertices and deleting them if they have degree more than $k+d$. Hence, it takes $O(|V(G)|+|E(G)|) = O(n(k+d))$ time in total in order to apply Reduction Rule 4.1.

Likewise, iterating over all vertices, checking the degree condition in Reduction Rule 4.2, and deleting the corresponding vertices, can be done in $O(n(k+d))$ time. □

In the following, assume that the instance (G, k) is reduced with respect to Reduction Rule 4.1 and Reduction Rule 4.2.

Lemma 4.2. *An instance that is reduced with respect to Reduction Rule 4.1 and Reduction Rule 4.2 contains at most $O(k(k+d)d)$ vertices.*

Proof. Assume that S is a bdd-d-set of size k. Due to Reduction Rule 4.1, each vertex has degree at most $d+k$. Therefore, $|N(S)| \leq k(d+k)$. Clearly, each vertex in $D := N(S)$ has degree at most d in $G - S$, because S is a bdd-d-set. Thus, for the vertices in $F := N(D) \setminus S$ we know that $|F| \leq k(d+k)d$. Due to Reduction Rule 4.2, there are no further neighbors of $F \setminus (C \cup D)$, that is, $S \cup D \cup F$ contains all vertices of the reduced instance, and we obtain the claimed bound $k + k(d+k) + k(d+k)d = O(k(k+d)d)$ on the number of vertices in the reduced instance. □

The following theorem follows immediately.

Theorem 4.1. Bounded-Degree-d Vertex Deletion *admits a problem kernel of $O(k(k+d)d)$ vertices, which can be constructed in $O(n(k+d))$ time.*

In the next section, we will see that a significantly better kernel size can be obtained.

4.3 A Local Optimization Algorithm

Nemhauser and Trotter [NT75] proved a famous theorem in combinatorial optimization. In terms of the NP-hard Vertex Cover problem, it can be formulated as follows:

NT-Theorem [NT75, BYE85]. *For an undirected graph $G = (V, E)$, one can compute in polynomial time two disjoint vertex subsets A and B such that the following three properties hold:*

1. *If S' is a vertex cover of the induced subgraph $G - (A \cup B)$, then $A \cup S'$ is a vertex cover of G.*
2. *There is a minimum-cardinality vertex cover S of G with $A \subseteq S$.*
3. *For every vertex cover S'' of the induced subgraph $G - (A \cup B)$,*

$$|S''| \geq \frac{|V \setminus (A \cup B)|}{2}.$$

In other words, the NT-Theorem provides a polynomial-time data reduction for Vertex Cover. That is, for vertices in A it can already be decided in polynomial time to put them into the solution set and vertices in B can be ignored for finding a solution. Hochbaum [Hoc82] first explained that the NT-Theorem is very useful for approximating Vertex Cover. The point is that the

search for an approximate solution can be restricted to the induced subgraph $G-(A\cup B)$. The NT-Theorem directly delivers a factor-2 approximation for Vertex Cover by choosing $V \setminus B$ as the vertex cover. Chen et al. [CKJ01] first observed that the NT-Theorem directly yields a $2k$-vertex problem kernel for Vertex Cover, where the parameter k denotes the size of the solution set. Indeed, this is in a sense an "ultimate" kernelization result in parameterized complexity analysis [DF99, FG06, Nie06] because there is good reason to believe that there is a matching lower bound $2k$ for the kernel size unless P = NP [KR08].

Since its publication numerous authors have referred to the importance of the NT-Theorem from the viewpoint of polynomial-time approximation algorithms (e.g., [BYE85, Khu02]) as well as from the viewpoint of parameterized algorithmics (e.g., [AFLS07, CKJ01, GN07a, CC08]). The relevance of the NT-Theorem comes from both its practical usefulness in solving Vertex Cover [ACF+04] as well as its theoretical depth having led to numerous further studies and follow-up work [AFLS07, BYE85, BYRH09, CC08].

In this section, our contribution is to provide a version of the NT-theorem that generalizes to Bounded-Degree-d Vertex Deletion. The corresponding algorithmic strategies and proof techniques, however, are not achieved by a generalization of known proofs of the NT-Theorem.

Theorem 4.2 (BDD-DR-Theorem)**.** *For an undirected graph $G = (V, E)$ and any constant $\epsilon > 0$, one can compute in $O(n^4 \cdot m)$ time two disjoint vertex subsets A and B such that the following three properties hold:*

1. *If S' is a bdd-d-set of the induced subgraph $G - (A \cup B)$, then $A \cup S'$ is a bdd-d-set of G.*
2. *There is a minimum-cardinality bdd-d-set S of G with $A \subseteq S$.*
3. *For every bdd-d-set S'' of the induced subgraph $G - (A \cup B)$, for $d \leq 1$,*

$$|S''| \geq \frac{|V \setminus (A \cup B)|}{d^3 + 4d^2 + 6d + 4}$$

and for $d \geq 2$,

$$|S''|^{1+\epsilon} \geq \frac{|V \setminus (A \cup B)|}{c}$$

for some constant c depending on d and ϵ.

There is a significant difference to the NT-theorem for Vertex Cover: for $d \geq 2$, with our technique we can only get arbitrarily close to a linear dependence of the minimum solution size on the number of vertices in $|V \setminus (A \cup B)|$. The first two properties of Theorem 4.2 are called the *local optimality conditions*, since they guarantee that we can "locally" decide to take all vertices in A into an optimal solution set and vertices in B can be ignored for finding a solution. The third property of Theorem 4.2 is called the *size condition*.

As a direct application of Theorem 4.2 we obtain a problem kernel (by simply removing $A \cup B$ from the graph and setting $k := k - |A|$):

Corollary 4.1. *For $d \leq 1$,* Bounded-Degree-d Vertex Deletion *admits a problem kernel of at most $(d^3 + 4d^2 + 6d + 4) \cdot k$ vertices, and for constant $d \geq 2$,* Bounded-Degree-d Vertex Deletion *admits a problem kernel of $O(k^{1+\epsilon})$ vertices for any constant $\epsilon > 0$. Both problem kernels can be computed in $O(n^4 \cdot m)$ time.*

We begin to describe the main algorithm that computes two sets A and B as claimed in Theorem 4.2.

4.3.1 The Main Algorithm

The first step to prove Theorem 4.2 is to greedily compute a factor-$(d+2)$ approximate bdd-d-set X for G. To this end, we use the following easy-to-verify forbidden subgraph characterization of bounded-degree graphs: A graph G has maximum degree d if and only if there is no $(d+1)$-star (a star with $d+1$ leaves) as a subgraph in G. With a straightforward greedy algorithm, compute a *maximal $(d+1)$-star packing* of G, that is, a set of vertex-disjoint $(d+1)$-stars that cannot be extended by adding another $(d+1)$-star. Let X be the set of vertices of this star packing. Since the number of stars in the packing is a lower bound for the size of a minimum bdd-d-set, X is a factor-$(d+2)$ approximate bdd-d-set. Greedily remove vertices from X such that X is still a bdd-d-set, and finally set $Y := V \setminus X$. These two vertex sets X and Y are the starting point for the search for the two vertex subsets A and B that fulfill the properties in Theorem 4.2; as we will see, we can restrict A to be a subset of X and B to be a subset of Y.

Since X is a factor-$(d+2)$ approximate bdd-d-set, it is clear that every bdd-d-set S'' contains at least $|X|/(d+2)$ vertices, that is, $|S''| \geq |X|/(d+2)$. Thus, $|X| \leq |S''| \cdot (d+2)$. Roughly speaking, this shows that the size condition (third property) of Theorem 4.2 is fulfilled by choosing $A := \emptyset$ and $B := Y = V \setminus X$. However, this choice of A and B will in general not guarantee that the first two properties of Theorem 4.2 are fulfilled. To fulfill the first two properties, only a subset of Y can be chosen to be contained in B, but this subset should be as large as possible in order to fulfill the size condition, that is, one has to bound the size of $Y \setminus B$ with respect to $|X|$. The most important lemma, whose proof is deferred to the next subsections, shows that if $Y = V \setminus X$ is too big compared to X, then one can find two vertex sets $A' \subseteq X$ and $B' \subseteq Y$ that fulfill the local optimality conditions such that B' is not empty.

Lemma 4.3. *Let $G = (V, E)$ be a graph with a bdd-d-set X. If $Y = V \setminus X$ contains more than $(d+1)^2 \cdot |X|$ vertices for $d \leq 1$ or more than $O(|X|^{1+\epsilon})$ vertices for $d \geq 2$, then one can find in $O(n^3 m)$ time two vertex subsets $A' \subseteq X$ and $B' \subseteq Y$ such that the following three properties hold:*

1. *If S' is a bdd-d-set of the induced subgraph $G - (A' \cup B')$, then $A' \cup S'$ is a bdd-d-set of G.*

Algorithm: ComputeAB (G)
Input: An undirected graph $G = (V, E)$.
Output: Vertex subsets A and B satisfying the three properties of Theorem 4.2.

```
1  A ← ∅, B ← ∅
2  Compute a (d + 2)-factor approximate bdd-d-set X for G.
3  Y ← V \ X
4  if d ≤ 1 then
5        if |Y| ≤ (d + 1)^2 · |X| then return (A, B)
6  if d ≥ 2 then
7        if |Y| ≤ c' · |X|^(1+ε) then return (A, B)
8  (A', B') ← FindExtremal (G, X).
9  G ← G − (A' ∪ B')
10 A ← A ∪ A'
11 B ← B ∪ B';
12 goto line 2
```

Figure 4.1: Pseudo-code of the main algorithm for computing A and B. The exact value of the constant c', which is depending on d and ϵ, is determined later in the proof of Proposition 4.4. The pseudo-code of FindExtremal is given in Figure 4.5.

2. There is a minimum-cardinality bdd-d-set S of G with $A' \subseteq S$.

3. The subset B' is not empty.

Note that the first two properties, that is, the local optimality conditions, are the same as the first two properties in Theorem 4.2. As the third property in Theorem 4.2, we also call the third property in Lemma 4.3 the *size condition*. The reason is as follows. The main algorithm iteratively applies the algorithm behind Lemma 4.3 and removes A' and B' from G and recomputes X, until the preconditions of Lemma 4.3 are not fulfilled anymore. The union of all A''s and B''s, respectively, then forms the sets A and B with the properties that are stated in Theorem 4.2. Since B' is never empty, we have a guarantee that B "grows bigger" in each iteration, which will eventually bound the size of $Y \setminus B$, and this bound will almost directly imply the size condition of Theorem 4.2.

In the following, we show the correctness of this approach. Let FindExtremal[1] be an algorithm that finds two subsets A' and B' as stated in Lemma 4.3. Figure 4.1 shows the pseudo-code of the main algorithm that will be used to show Theorem 4.2. The algorithm starts initializing A and B with empty sets in line 1, and then it computes a factor-$(d+2)$ approximate bdd-d-set X in line 2 and the

[1]The name "FindExtremal" comes from extremal combinatorics arguments that we use in the proof. Such arguments are often used for parameterized algorithms and problem kernelization, see, e.g., [ECFLR05].

remaining vertices Y in line 3. If the set Y is small compared to X (conditions in line 5 or line 7), then (A, B) is returned. If the set Y is too big compared to X (that is, the conditions in line 5 or line 7 are not fulfilled), then, in line 8, the graph G and the vertex set X are passed to the procedure FindExtremal, which computes two sets A' and B' satisfying the properties in Lemma 4.3. The sets A' and B' are then added to A and B in lines 10 and 11, respectively. Finally, in line 12 the algorithm starts over and computes a factor-$(d+2)$ approximate solution for the new graph G in line 2. Since B' is never empty due to Lemma 4.3, the conditions in line 5 or line 7 will eventually be fulfilled (because in each iteration at least one vertex is added to B, thus eventually $Y \setminus B$ will be "small"), and the algorithm returns the vertex subsets A and B. It remains to show that A and B fulfill the three properties of Theorem 4.2, and that the running time of ComputeAB is $O(n^4 \cdot m)$.

Lemma 4.4. *The sets A and B computed by* ComputeAB *fulfill the three properties given in Theorem 4.2.*

Proof. First, we prove that A and B fulfill the first two properties of Theorem 4.2. The proof is by a simple inductive argument: assume that in some iteration of ComputeAB the two vertex subsets A and B of a graph G fulfill the first two properties (local optimality conditions) of Theorem 4.2 with respect to G (call this assumption a), and that FindExtremal returns in line 8 two vertex subsets A' and B' of $G' := G - (A \cup B)$ that fulfill the first two properties (local optimality conditions) of Lemma 4.3 with respect to the graph G' (call this assumption b). We show that then $A \cup A'$ and $B \cup B'$ fulfill the first two properties of Theorem 4.2 with respect to G as well:

1. If S' is a bdd-d-set of the induced subgraph $G' - (A' \cup B') = G - (A \cup A' \cup B \cup B')$, then $A' \cup S'$ is a bdd-d-set of G' (due to assumption b), and therefore $A \cup A' \cup S'$ is a bdd-d-set of G (due to assumption a). This shows the first property.

2. There is a minimum-cardinality bdd-d-set S of G with $A \subseteq S$ (due to assumption a). Since the graph property "bounded degree d" is hereditary, $S \setminus (A \cup B)$ is a bdd-d-set for $G' = G - (A \cup B)$. There is a minimum-cardinality bdd-d-set S' of G' with $A' \subseteq S'$ (due to assumption b). Since S' has minimum cardinality, $|S'| \leq |S \setminus (A \cup B)|$. The set $S' \cup A$ is a bdd-d-set of G (due to assumption a), and because $A \subseteq S$ we know that $|S' \cup A| \leq |S|$. Since S has minimum cardinality, $S' \cup A$ has minimum-cardinality, and thus $S' \cup A$ is a minimum-cardinality bdd-d-set that contains $A' \cup A$. This shows the second property.

The sets $A = \emptyset$ and $B = \emptyset$ (line 1) trivially fulfill the first two properties of Theorem 4.2, and by the above inductive argument the sets A and B returned by ComputeAB fulfill these properties as well.

It remains to show that the sets A and B fulfill the third property.

3. Let $V' := V \setminus (A \cup B)$. Since the condition in line 5 (for $d \leq 1$) or line 7 (for $d \geq 2$) is true, we know that either $|Y| \leq (d+1)^2 \cdot |X|$ and therefore

$$|V'| = |X| + |Y| \leq (1 + (d+1)^2) \cdot |X| \quad \text{(for } d \leq 1\text{), or}$$

$|Y| = O(|X|^{1+\epsilon})$ and therefore

$$|V'| = |X| + |Y| = O(|X|^{1+\epsilon}) \quad \text{(for } d \geq 2\text{).}$$

Recall that X is a factor-$(d+2)$ approximate bdd-d-set for $G' := G-(A \cup B)$. Thus, $|X| \leq (d+2) \cdot |S|$ for an arbitrary bdd-d-set S.

For $d \leq 1$, one obtains $|V'| \leq (1+(d+1)^2) \cdot |X| \leq (1+(d+1)^2)(d+2) \cdot |S|$ and therefore

$$|S| \geq \frac{|V'|}{(1+(d+1)^2)(d+2)} = \frac{|V'|}{d^3+4d^2+6d+4}.$$

For $d \geq 2$, one obtains $|V'| = O(|X|^{1+\epsilon}) = O(|S|^{1+\epsilon})$ and therefore

$$|S|^{1+\epsilon} \geq \frac{|V'|}{c}$$

for some constant c. This shows the third property.

□

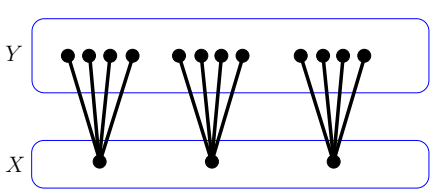
Next, we show the running time of ComputeAB.

Lemma 4.5. *Algorithm* ComputeAB *runs in* $O(n^4 \cdot m)$ *time.*

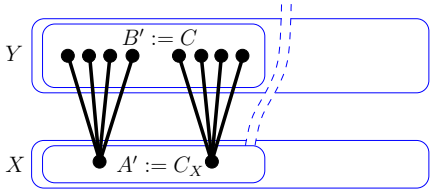
Proof. With the described simple greedy approach, computing a factor-$(d+2)$ approximate solution in line 2 takes $O(n+m)$ time. Each call of FindExtremal in line 8 takes $O(n^3 \cdot m)$ time. FindExtremal always returns two sets A' and B' such that B' is not empty (Lemma 4.3), hence after at most n iterations of ComputeAB, Y must be small compared to X and ComputeAB returns in line 5 (for $d \leq 1$) or line 7 (for $d \geq 2$). Thus, in total, we find the sets A and B in $O(n^4 \cdot m)$ time. □

With Lemmas 4.4 and 4.5, the proof of Theorem 4.2 is completed.

The remaining part of this section is dedicated to the proof of Lemma 4.3 by providing a description of the algorithm FindExtremal. The description is divided into an outline (Section 4.3.2), the description of a method to show the local optimization conditions (Section 4.3.3), the description of an important subroutine (Section 4.3.4), and finally a description of FindExtremal together with the correctness proofs (Section 4.3.5).



(a) Each vertex in X is the center of a star with four leaves in Y.



(b) Each vertex in A' is the center of a star with four leaves in $B' \subseteq Y$. The dashed lines illustrate that there are no edges between B' and the rest of the graph in $G - A'$.

Figure 4.2: Some simple cases for FindExtremal, assuming $d = 3$.

4.3.2 The Ingredients of FindExtremal

We prove Lemma 4.3 by describing an algorithm called FindExtremal that, given an undirected graph G and a bdd-d-set X such that $Y := V \setminus X$ is "large" compared to X, finds two subsets $A' \subseteq X$ and $B' \subseteq Y$ such that they fulfill the local optimality conditions and such that B' is not empty (see Lemma 4.3). We first focus on the local optimality conditions:

1. If S' is a bdd-d-set of the induced subgraph $G - (A' \cup B')$, then $A' \cup S'$ is a bdd-d-set of G.

2. There is a minimum-cardinality bdd-d-set S of G with $A' \subseteq S$.

Informally speaking, these two properties guarantee that one can always assume that there exists a minimum-cardinality bdd-d-set that contains all vertices in A' and no vertex in B'. We use this informal interpretation for the following step-by-step explanation of the main obstacles that FindExtremal has to bypass.

How to Fulfill the Local Optimality Conditions. The fundamental idea to show the local optimality conditions is to use the forbidden subgraph characterization of bounded-degree graphs: a graph G has maximum degree d if and only if there is no $(d+1)$-star (a star with $d+1$ leaves) as a subgraph in G. To illustrate the idea, let us first assume that there is a packing of vertex-disjoint $(d+1)$-stars in G such that each vertex in the bdd-d-set X is the center of such a star (thus, all leaves are in Y). Hence, each vertex in X is "covered" by the star packing. See Figure 4.2a for an example. Then, due to the forbidden subgraph characterization, a minimum-cardinality bdd-d-set has to contain at least one vertex of each star, thus a minimum-cardinality bdd-d-set contains at least $|X|$ vertices, and X (that is, the set of all centers) is therefore a minimum-cardinality bdd-d-set of G. Thus, for $A' := X$ and $B' := Y$ there exists a minimum-cardinality bdd-d-set that contains all vertices in A' and no vertex in B'.

Obviously, in general there might not exist a vertex-disjoint packing of $(d+1)$-stars whose centers cover all vertices in X; rather, it can happen that one is only

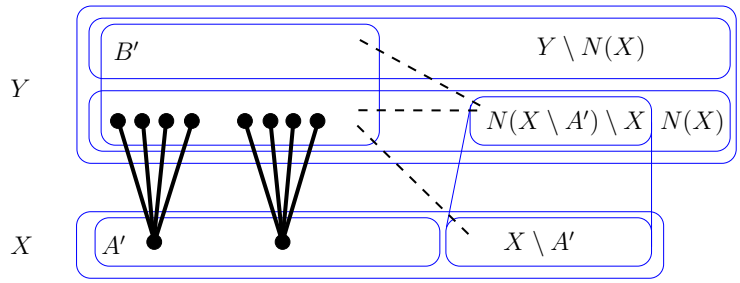


Figure 4.3: Illustration of the structure of the graph with bdd-d-set X (assuming $d = 3$), its neighborhood $N(X)$, and all remaining vertices $Y \setminus N(X)$. The sets A' and B' fulfill the A'-star cover property and the restricted neighborhood properties, because there is a $(d+1)$-star with center in A' and four leaves in B' for each vertex in A' and there are no edges between B' and $X \setminus A'$ and no edges between B' and $N(X \setminus A') \setminus X$ (illustrated by dashed lines).

able to find a subset C_X of X whose vertices are centers of $(d+1)$-stars with leaves in Y. Now suppose that the subset $C_X \subseteq X$ is a separator in G such that the leaves of these $(d+1)$-stars are contained in a component C that is "separated" from the rest of the graph, that is, every path from C to a vertex neither in C nor in C_X passes through C_X. See Figure 4.2b for an example. Then, due to the forbidden subgraph characterization, a minimum-cardinality bdd-d-set has to contain at least one vertex for each $(d+1)$-star with center in C_X, and taking all vertices in C_X into a solution is always optimal, since each vertex in C has degree at most d in $G - C_X$. Thus, for $A' := C_X$ and $B' := V(C)$ there exists a minimum-cardinality bdd-d-set that contains all vertices in A' and no vertex in B'. If C_X is not a separator as described, then this approach does not work directly; however, as we will see, it is not necessary that A' is a separator that completely separates B' from the rest of the graph; A' and B' only have to fulfill the following three properties:

A'-star cover property: There exists a packing of vertex-disjoint stars in $G[A' \cup B']$, each star having at least $d+1$ leaves, such that each vertex in A' is the center of such a star.

Restricted X-neighborhood property: There are no edges between B' and $X \setminus A'$.

Restricted Y-neighborhood property: There are no edges between B' and $N(X \setminus A') \setminus X$.

See Figure 4.3 for an illustration. Note the difference to the case illustrated in Figure 4.2b: with these three properties, the set A' is not necessarily a separator. Intuitively, the A'-star cover property is needed to prove that there exists an optimal bdd-d-set containing A', the restricted X-neighborhood property is needed

to avoid that a vertex in B' has degree more than d in $G - A'$, and the restricted Y-neighborhood property is needed to avoid that a *neighbor* of a vertex in B' has degree more than d in $G - A'$ (because, since X is a bdd-d-set of G, the only vertices of degree more than d in $G - A'$ can be in $X \setminus A'$ or in $N(X \setminus A') \setminus X$). Roughly speaking, then, similarly to the ideas outlined above, it is always optimal to take all vertices in A' into the solution, and the vertices in B' do not have to be considered for an optimal solution, since they and their neighbors have degree at most d in $G - A'$; the formal correctness proof is given in Section 4.3.3. With the A'-star cover property and the restricted neighborhood properties we are now ready to sketch how FindExtremal works.

FindExtremal in a Nutshell. The algorithm FindExtremal guarantees the A'-star cover property and the restricted X-neighborhood property as follows. FindExtremal computes a packing of $(d+1)$-stars between X and Y such that the centers C_X of the stars are in X and the leaves in Y, and such that the leaves of the stars are not adjacent to vertices in $X \setminus C_X$. This is accomplished by a procedure called StarPacking based on maximum flow techniques, which is described in detail in Section 4.3.4. Roughly speaking, by setting $A' := C_X$ and by choosing B' such that it contains all leaves of the $(d+1)$-stars, A' and B' fulfill the A'-star cover and the restricted X-neighborhood property. The more difficult part is to fulfill the restricted Y-neighborhood property. There might be edges between leaves of the $(d+1)$-stars and vertices in $N(X \setminus A') \setminus X$. If there are no such edges, then the A'-star cover and restricted neighborhood properties are fulfilled. If there are such edges, then the trick is to "forbid" the vertices in $X \setminus A'$, $N(X \setminus A') \setminus X$, and their neighbors in Y (that is, all vertices in Y within distance at most two in $G - A'$ to a vertex in $X \setminus A'$). These forbidden vertices will contain some leaves of the packing, thus the star packing between X and Y is recomputed, but excluding the forbidden vertices. The point is, as we will see, that if the recomputed packing can be used to construct A' and B' fulfilling the A'-star cover property and the two restricted neighborhood properties in the subgraph excluding the forbidden vertices, then they also fulfill these properties in G. This approach of computing a packing and forbidding vertices is then iterated until the algorithm finds two vertex subsets A' and B' fulfilling the local optimality conditions. In summary, FindExtremal works roughly as follows:

1. Call StarPacking to compute a packing of $(d+1)$-stars (in order to fulfill the A'-star cover property and the restricted X-neighborhood property), excluding forbidden vertices.

2. If the restricted Y-neighborhood property can be fulfilled, then construct A' and B' and return.

3. Forbid vertices that prevent that the restricted Y-neighborhood property can be fulfilled.

4. Goto 1

Of course it has to be shown in detail that this approach always terminates and returns a correct solution. Moreover, it still remains to consider the size condition of Lemma 4.3.

How to Fulfill the Size Condition. Next, consider the size condition of Lemma 4.3:

> The subset B' is not empty.

Recall the preconditions of Lemma 4.3: FindExtremal is only called if $Y := V \setminus X$ contains more than $(d+1)^2 \cdot |X|$ vertices for $d \leq 1$ or more than $O(|X|^{1+\epsilon})$ vertices for $d \geq 2$. The problem is that in the iterative process of computing a star packing and forbidding vertices, all vertices in X and Y might become forbidden, and hence FindExtremal would not be able to return a non-empty vertex subset B'. However, one can show that not too many vertices become forbidden in this process, and that there will be always some vertices in B' left. To make this possible, it is necessary that the StarPacking procedure finds a star packing such that the set C_X of centers of $(d+1)$-stars is "as large as possible" and that the remaining vertices in $X \setminus C_X$ have only few neighbors in Y. Then, roughly speaking, since $X \setminus C_X$ contains few vertices, there are only few neighbors $N(X \setminus C_X) \setminus X$, and since X is a bdd-d-set, each vertex in $N(X \setminus C_X) \setminus X$ has only d neighbors in Y. Hence, there are only few forbidden vertices, and summing up the number of all forbidden vertices over all iterations will show that there can be only $(d+1)^2 \cdot |X|$ forbidden vertices in Y for $d \leq 1$ or $O(|X|^{1+\epsilon})$ forbidden vertices for $d \geq 2$.

Remarks. Note that the above description of FindExtremal is somewhat simplified; for the case $d \leq 1$ it works as described, but for the case $d \geq 2$ we actually compute a packing of stars with more than $d+1$ leaves. The adapted number of leaves depends on ϵ and $|X|$ and guarantees that FindExtremal iterates only a constant number of times (where the constant is depending on ϵ). Moreover, for $d \geq 2$ it is possible that the algorithm returns two subsets A' and B' that do not fulfill the A'-star cover and restricted neighborhood properties but nevertheless the local optimality conditions. However, we emphasize that the main concept is the same as for $d \leq 1$, the difference becomes important in the formal proof of the correctness of FindExtremal.

Proof Structure of the Remainder of this Section. In the following, we shortly outline the structure of the remaining description of FindExtremal. The main parts are as follows.

1. A proof that if A' and B' fulfill the A'-star cover property and the restricted neighborhood properties, then they also fulfill the local optimality conditions (Section 4.3.3).

2. A description of the StarPacking algorithm (used by FindExtremal in order to fulfill the A'-star cover property and the restricted X-neighborhood property) and its correctness proof (Section 4.3.4).

3. A description of the FindExtremal algorithm and its correctness proof (Section 4.3.5). This part is organized as follows, roughly ordered by increasing technical difficulty.

   (a) A pseudo-code formulation of FindExtremal.

   (b) Proof of the running time of FindExtremal.

   (c) Proof that FindExtremal always outputs A' and B' fulfilling the local optimality conditions.

   (d) Proof of the size condition, that is, that FindExtremal always returns a nonempty set B'.

4.3.3 Star Cover and Restricted Neighborhood Properties

Recall that FindExtremal tries to find two subsets A' and B' that fulfill the local optimality conditions and the size condition. We repeat the definition of the A'-star cover property and the restricted neighborhood properties, and show that these properties suffice to show the local optimality conditions.

A'-star cover property: There exists a packing of vertex-disjoint stars in $G[A' \cup B']$, each star having at least $d+1$ leaves, such that each vertex in A' is the center of such a star.

Restricted X-neighborhood property: There are no edges between B' and $X \setminus A'$.

Restricted Y-neighborhood property: There are no edges between B' and $N(X \setminus A') \setminus X$.

See Figure 4.3 for an illustration of these properties. First, we show that these three properties are at least as strong as the local optimality conditions, that is, the first two properties of Lemma 4.3:

Lemma 4.6. *Let A' and B' be two vertex subsets satisfying the A'-star cover property and the restricted neighborhood properties. Then, the following two properties hold:*

(1) If S' is a bdd-d-set of the induced subgraph $G - (A' \cup B')$, then $A' \cup S'$ is a bdd-d-set of G.
(2) There is a minimum-cardinality bdd-d-set S of G with $A' \subseteq S$.

Proof. To prove (1), suppose that S' is a bdd-d-set of $G' := G - (A' \cup B')$. To prove that $S'' := S' \cup A'$ is a bdd-d-set of G, we have to consider the vertices in $N_G[B'] \setminus S''$. For these vertices we have to show that their degree is at most d in $G - S''$, that is, each vertex in $N_G[B'] \setminus S''$ has degree at most d in $G - S''$. To this end, we show that each vertex in $N_G[B'] \setminus A' \supseteq N_G[B'] \setminus S''$ has degree at most d in $G - A'$. Since X is a bdd-d-set of G and since $A' \subseteq X$, the only vertices that can have degree more than d in $G - A'$ are in $X \setminus A'$ and in $N(X \setminus A') \setminus X$, but these vertices are neither in B' nor are they neighbors of vertices in B' due to the restricted neighborhood properties, and hence each vertex in $N_G[B'] \setminus A'$ has degree at most d in $G - A'$ and, therefore, also in $G - S''$.

Before proving (2), one needs to show that A' is a minimum-cardinality bdd-d-set of $G[A' \cup B']$. Since X is a bdd-d-set of G, the vertex subset A' is a bdd-d-set of $G[A' \cup B']$; moreover, due to the A'-star cover property, for each vertex $v \in A'$ there is a star with at least $d + 1$ leaves in B' with center v. Since each star has at least $d + 1$ leaves, it has to contain at least one vertex of a minimum-cardinality bdd-d-set of $G[A' \cup B']$, and, therefore, every bdd-d-set of $G[A' \cup B']$ contains at least $|A'|$ vertices, showing that A' is a minimum-cardinality bdd-d-set of $G[A' \cup B']$.

To prove (2), suppose that S' is a minimum-cardinality bdd-d-set of G. If $A' \subseteq S'$, then we are done. Therefore, assume that $A' \not\subseteq S'$. We show that we can transform S' into a bdd-d-set S with $|S| = |S'|$ and $A' \subseteq S$. Let $A'' := A' \setminus S'$. As shown above, the set A' is a minimum-cardinality bdd-d-set of $G[A' \cup B']$. Since the bounded-degree property is hereditary, $S' \cap (A' \cup B')$ is a bdd-d-set of $G[A' \cup B']$. Since A' is a minimum-cardinality bdd-d-set of $G[A' \cup B']$, for the vertex subset $B'' := B' \cap S'$ we know that $|A''| \leq |B''|$. We claim that the set $S := (S' \setminus B'') \cup A''$ (thus, $A' \subseteq S$) is also a bdd-d-set of G. Since the vertices in B'' are the only vertices in $S' \setminus S$, it suffices to show that these vertices and their neighbors have degree at most d in $G - S$. As shown in the proof of (1), each vertex in $N_G[B'] \setminus A'$ has degree at most d in $G - A'$ and thus each vertex in $N_G[B''] \setminus A'$ has degree at most d in $G - A'$ and, therefore, S is a bdd-d-set of G. Due to $|A''| \leq |B''|$, S is a minimum-cardinality bdd-d-set of G with $A' \subseteq S$. □

Lemma 4.6 will be used in the proof of the correctness of FindExtremal—it helps to make the description of the underlying algorithm and the corresponding correctness proofs more accessible.

As described in the outline of FindExtremal (Section 4.3.2), the search for the subsets $A' \subseteq X$ and $B' \subseteq Y$ will be driven by the search for a packing of vertex-disjoint stars with centers in X and at least $(d + 1)$ leaves in $N(X)$. Roughly speaking, the centers of such stars with at least $d+1$ leaves will be in A' and their leaves will be in B', which fulfills the A'-star cover property, but in order to fulfill the restricted neighborhood properties the vertices for A' and B' have to be selected carefully. To fulfill the third property of Lemma 4.3, which says that the returned vertex set $B' \subseteq Y$ is not empty, it is necessary that the packing of stars with centers in X and leaves in $N(X)$, which is used to compute A' and B',

contains "as many stars with at least $d+1$ leaves as possible". This is described in more detail in the next subsection.

4.3.4 Star Packing

As outlined in the preceding subsections, given a graph G and a bdd-d-set X, the task is to compute a star packing P with the centers of the stars being from X and the leaves being from $N(X) \subseteq Y = V \setminus X$. The stars in the packing shall have at most r leaves, where r depends on d and is set to

$$r := \begin{cases} d+1, & \text{if } d \leq 1, \\ d+1+\lceil |X|^{\epsilon} \rceil, & \text{otherwise.} \end{cases}$$

The reason for this distinction between $d \leq 1$ and $d \geq 2$ will become clear later in the analysis of the algorithm. For the moment it is only important that $r \geq d+1$, which implies that an r-star is a forbidden subgraph. To compute the star packing P, we relax, on the one hand, the requirement that the stars in the packing have exactly r leaves, that is, the packing P might contain $<r$-stars. On the other hand, P shall have a maximum number of edges. The rough idea behind this requirement for a maximum number of edges is to maximize the number of r-stars in P, and to guarantee that the leaves of many r-stars are not adjacent to centers of $<r$-stars. Based on P, it is possible to "separate" many r-stars, whose centers will be in A' and whose leaves will be in B', such that their leaves are not adjacent to the center of any $<r$-star. This will guarantee that there are no edges between B' and $X \setminus A'$ (restricted X-neighborhood property). For computing such a star packing, we can restrict our attention to the bipartite graph J induced by the edges between X and $N(X)$, that is, $V(J) = X \cup N(X)$ and $E(J) = \{\{u,v\} \in E(G) \mid u \in X \text{ and } v \in N(X)\}$. The following lemma is a precise statement of the properties of the star packing. In the lemma, the centers of the "separated" r-stars are contained in a vertex set $C_X \subseteq X$ and the leaves of the remaining stars are contained in a vertex set $C_Y \subseteq Y$. The fact that there are no edges between the leaves of the "separated" r-stars and the centers of the remaining stars is expressed by saying that $C_X \cup C_Y$ is a vertex cover in J. Since FindExtremal will call the star packing algorithm for subsets of X and Y, we state the lemma with respect to $X' \subseteq X$ and $Y' \subseteq Y$.

Lemma 4.7. *In a bipartite graph J with vertex sets X' and Y', one can find in $O(n^2 \cdot m)$ time a $\leq r$-star packing P and a vertex cover $C_X \cup C_Y$ of J, where $C_X \subseteq X'$ and $C_Y \subseteq Y'$ such that*

1. *every vertex of C_X is the center of an r-star in P and the leaves of the r-stars in P (with center in C_X) are not in C_Y, and*
2. *every vertex of C_Y is a leaf in the star packing (of some $\leq r$-star with center in $X' \setminus C_X$).*

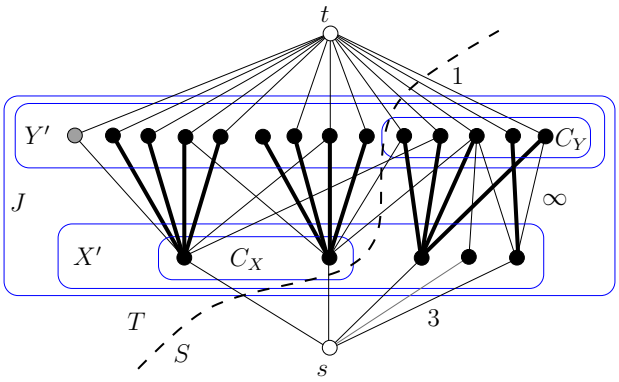


Figure 4.4: Example of a bipartite graph J, a packing P (bold edges and black vertices) in J that fulfills the properties stated in Lemma 4.7 for $r = 4$, and the additions to the graph that are used to prove Lemma 4.7 by maximum flow / minimum cut duality (s, t, and their incident edges). The corresponding flow network has the source s and the sink t (assuming that all edges are directed from bottom to top), where each edge incident to s has capacity $r = 4$, each edge in J has capacity ∞, and each edge incident to t has capacity 1. The dashed line shows a minimum s-t-cut (S, T), which can be used to compute C_X and C_Y.

See Figure 4.4 for an example of such a packing P with vertex cover $C_X \cup C_Y$. Let ComputePacking(J,X',Y') be an algorithm that computes such a packing P and two vertex subsets C_X and C_Y as stated in Lemma 4.7.

Proof. From the given bipartite graph J, construct a flow network as follows (see Figure 4.4). Introduce two new vertices s and t, and add an edge with capacity r from s to v for every $v \in X'$, add an edge with capacity 1 from w to t for every $w \in Y'$, and add an edge with infinite capacity from $v \in X'$ to $w \in Y'$ if $\{v, w\} \in E(J)$. A maximum flow f corresponds to a packing P of $\leq r$-stars in J. Let (S, T) be the corresponding cut of capacity f with $s \in S$ and $t \in T$. The set $C_X \cup C_Y$ with $C_X := X' \cap T$ and $C_Y := Y' \cap S$ is a vertex cover for J; otherwise, there would be an edge with infinite capacity that leaves S, contradicting the fact that (S, T) has capacity f. Moreover, one can observe that the vertices in C_X must be centers of vertex-disjoint r-stars, whose leaves are in T, and the vertices in C_Y must be leaves of stars in the corresponding packing P (otherwise, the cut (S, T) would have higher capacity than the maximum flow).

A maximum flow can be computed in $O(n^2 \cdot m)$ time using, e.g., "Dinic's algorithm" (cf. [Din06]). □

We mention in passing that the structure that is found by Lemma 4.7 can be interpreted as a generalization of *crown structures* for Vertex Cover (cf. [CFJ04, AFLS07]) to Bounded-Degree-d Vertex Deletion.

The algorithm ComputePacking is used by FindExtremal to find A'

and B' that fulfill the local optimality conditions. The FindExtremal algorithm is described next.

4.3.5 The FindExtremal Algorithm

As described in the outline of FindExtremal (Section 4.3.2), the approach of the FindExtremal algorithm is to iteratively call ComputePacking and use the returned packing P and the vertex sets C_X and C_Y to try to obtain two sets A' and B' satisfying the A'-star cover property and the restricted neighborhood properties, or, if that fails, to forbid parts of graph and to try again. An important addition for $d \geq 2$, which has not been mentioned so far, is that if FindExtremal fails too many times to find A' and B' satisfying the A'-star cover property and the restricted neighborhood property, then one can directly return two vertex sets A' and B' satisfying the local optimality conditions (the first two properties of Lemma 4.3).

The description of FindExtremal is divided into four parts. First, we give a pseudo-code description of FindExtremal, implementing the "trial-and-error" strategy outlined above. Then, we show the running time of FindExtremal. After that, we show that FindExtremal is correct in the sense that it returns two subsets A' and B' that fulfill the local optimality conditions of Lemma 4.3. For the output of FindExtremal there are two different cases to consider, for one of them we show that the A'-star cover property and the restricted neighborhood properties are fulfilled, and for the other case we directly show the validity of the local optimality conditions. Note that the first case applies for all $d \geq 0$, but the latter only applies for $d \geq 2$. Finally, we address the last property in Lemma 4.3, that is, we show that the vertex subset B' is never empty.

Pseudo-Code for FindExtremal

The pseudo-code in Figure 4.5 shows the algorithm FindExtremal. The input to FindExtremal is a graph G and a bdd-d-set X. It starts with computing the bipartite graph J induced by the edges between X and $N(X)$ (line 1). Vertices in X that are forbidden in the course of the algorithm execution are stored in the set F_X, which is initialized with an empty set (line 2). Vertices in Y that are forbidden in the course of the algorithm execution are stored in the set F_Y, which is also initialized with an empty set (line 2). The variable j counts the number of calls of ComputePacking (line 4) and is initialized with "1" (line 3). The algorithm always returns a vertex pair (A', B'), where $A' = X \setminus F_X$ and $B' = Y \setminus F_Y$ (lines 5 and 7), that is, it returns all the vertices that are not forbidden. There are two possible cases when (A', B') is returned:

1. either the vertex set C_X contains all vertices in X that are not in F_X (line 5) or

2. the algorithm has iterated $\lceil 1/\epsilon \rceil + 1$ times (line 7).

Procedure: FindExtremal (G, X)
Input: An undirected graph G and a bdd-d-set X of G.
Output: A vertex subset pair (A', B') satisfying the local optimality conditions.

```
1 J ← bipartite graph with X and N(X) as its two vertex subsets and
        E(J) ← {{u, v} ∈ E(G) | u ∈ X and v ∈ N(X)}
2 F_X ← ∅; F_Y ← ∅
3 j ← 1
4 (P, C_X, C_Y) ← ComputePacking(J − (F_X ∪ F_Y), X \ F_X, Y \ F_Y)
5 if C_X = X \ F_X then return(X \ F_X, Y \ F_Y)
6 F_X ← X \ C_X; F_Y ← N_G[N_J(F_X)] \ X
7 if d ≥ 2 and j ≥ ⌈1/ε⌉ + 1 then return(X \ F_X, Y \ F_Y)
8 j ← j + 1
9 goto 4
```

Figure 4.5: Pseudo-code of FindExtremal.

We will show that for each of these cases the pair (A', B') fulfills the local optimality conditions. If the first case does not apply, then the algorithm computes the new set F_X of forbidden vertices in X and updates the set F_Y of forbidden vertices in Y (line 6). After that, it is checked whether the second case applies (line 7). If not, the counter j is increased by one (line 8) and the algorithm starts over (line 9) by recomputing the star packing in line 4. See Figure 4.6 for an example of how FindExtremal works.

In the remainder of this section, we will show the following three statements.

Statement 4.1. *Algorithm* FindExtremal *in Figure 4.5 runs in* $O(n^3 \cdot m)$ *time.*

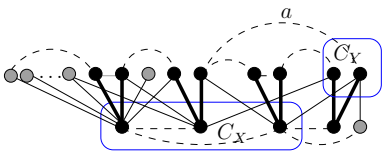
Statement 4.2. *Algorithm* FindExtremal *in Figure 4.5 returns two vertex subsets* (A', B') *that fulfill the local optimality conditions:*

1. *If* S' *is a bdd-d-set of the induced subgraph* $G - (A' \cup B')$*, then* $A' \cup S'$ *is a bdd-d-set of* G*.*

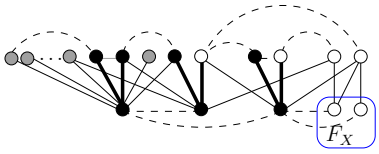
2. *There is a minimum-cardinality bdd-d-set* S *of* G *with* $A' \subseteq S$*.*

Statement 4.3. *Let* $G = (V, E)$ *be an undirected graph and let* X *be a bdd-d-set of* G*. If* $Y = V \setminus X$ *contains more than* $(d+1)^2 \cdot |X|$ *vertices for* $d \leq 1$ *or more than* $c' \cdot |X|^{1+\epsilon}$ *vertices for* $d \geq 2$ *(for some* c' *depending on* d *and* ϵ*), then algorithm* FindExtremal *in Figure 4.5 returns two vertex subsets* (A', B') *such that* B' *is not empty.*

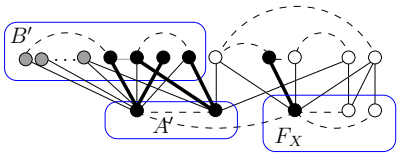
With Statements 4.1, 4.2, and 4.3, Lemma 4.3 follows immediately.



(a) An initial star packing P (bold edges) computed in line 4 with no vertex being forbidden. The A'-star cover property and the restricted X-neighborhood property could be fulfilled by choosing $A' = C_X$ and $B' = Y \setminus C_Y$. However, the restricted Y-neighborhood property cannot be fulfilled due to, for example, the edge a.



(b) Forbidding vertices. In line 6, the set F_X is computed, which is used to compute the vertices of Y excluded from the next iteration, namely, the set F_Y.



(c) Computing a new star packing P and F_X, and, then, two sets A' and B' that fulfill the A'-star cover property and the restricted neighborhood properties.

Figure 4.6: Example for $d = 1$ of how the algorithm FindExtremal in Figure 4.5 finds the two vertex subsets A' and B'. Solid lines (bold and non-bold) denote the edges in J, dashed lines denote the edges in $E(G) \setminus E(J)$. White vertices are forbidden and excluded from further iterations for computing the star packing P. Black vertices and bold edges are in the star packing P.

It remains to show the statements. First, we show the running time of FindExtremal (Statement 4.1). After that, we show that the returned sets A' and B' always fulfill the local optimality conditions (Statement 4.2). Finally, we show that the set B' is never empty (Statement 4.3).

Running Time of FindExtremal

In order to show the running time, we have to show that an updated F_X (line 6) is always a superset of an old one.

Lemma 4.8. *Assume that* FindExtremal *does not return in line 5, that is, $C_X \neq X \setminus F_X$. Let F'_X be the set $X \setminus C_X$ computed in line 6. Then it holds that $F_X \subsetneq F'_X$.*

Proof. In line 4 of FindExtremal, the packing P and the vertex sets C_X and C_Y are computed for the bipartite subgraph $J' = J - (F_X \cup F_Y)$ with $X' := X \setminus F_X$

and $Y' := Y \setminus F_Y$ as the two vertex subsets. Thus, since $C_X \neq X \setminus F_X = X'$, $C_X \subsetneq X'$, and, therefore, $F'_X = X \setminus C_X \supsetneq X \setminus X' = F_X$. □

Lemma 4.8 shows that FindExtremal will eventually terminate (for all $d \geq 0$), and we can prove the running time, which shows Statement 4.1.

Proposition 4.1. *Algorithm* FindExtremal *in Figure 4.5 runs in* $O(n^3 \cdot m)$ *time.*

Proof. In each iteration FindExtremal either returns in line 5 or line 7, or at least one vertex from X is added to F_X due to Lemma 4.8. Thus, after at most $|X| < n$ iterations, $F_X = X$. Then, $X \setminus F_X$ is empty and hence C_X returned by ComputePacking is empty, and, therefore, the condition $C_X = X \setminus F_X = \emptyset$ is true and FindExtremal returns in line 5. In each iteration, FindExtremal calls ComputePacking, which runs in $O(n^2 \cdot m)$ time (Lemma 4.7); all the other operations are simple assignments, if-instructions, and neighborhood computations, which take $O(n + m)$ time. □

Next, we show that the sets A' and B' returned by FindExtremal fulfill the local optimality conditions (Statement 4.2).

Fulfillment of Local Optimality Conditions

In order to show Statement 4.2, that is, that FindExtremal returns two vertex subsets (A', B') fulfilling the local optimality conditions, it is important to note that

$$F_Y = N_G[N_J(F_X)] \setminus X$$

is an invariant of FindExtremal (see line 6). We will need this invariant for the proofs of several lemmas and propositions. The next proposition corresponds to the case that FindExtremal returns in line 5.

Proposition 4.2. *Let* (P, C_X, C_Y) *be the output of* ComputePacking$(J-(F_X \cup F_Y), X \setminus F_X, Y \setminus F_Y)$ *(line 4). If* $C_X = X \setminus F_X$*, then* $A' = X \setminus F_X$ *and* $B' = Y \setminus F_Y$ *fulfill the local optimality conditions.*

Proof. We show that A' and B' fulfill the A'-star cover property and the restricted neighborhood properties. Due to Lemma 4.6, the sets A' and B' then also fulfill the local optimality conditions.

In line 4 of FindExtremal, the packing P and the vertex sets C_X and C_Y are computed for the bipartite subgraph $J' = J - (F_X \cup F_Y)$ with $X' := X \setminus F_X$ and $Y' := Y \setminus F_Y$ as the two vertex sets of J'. Thus, since $C_X = X'$, due to Lemma 4.7 we know that $C_Y = \emptyset$, since there are no stars with centers in $X' \setminus C_X = \emptyset$. Hence, due to Lemma 4.7 the vertices in C_X are centers of r-stars with leaves in Y', thus $A' = X'$ and $B' = Y'$ fulfill the A'-star cover property. We emphasize that this is also correct for the case $F_X = X$; in this case, the set A' is empty, and

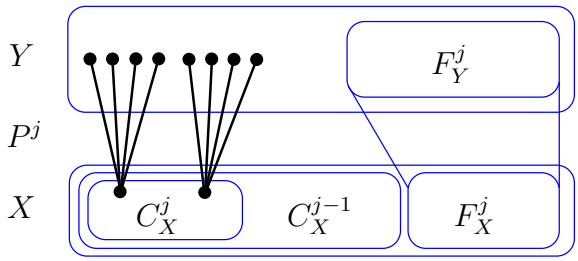


Figure 4.7: Illustration of the relation of the sets F_X^j, F_Y^j, C_X^j, C_X^{j-1}, and a packing P^j for $r = 4$. Only the r-stars with center in C_X^j in P_j are shown (there might exist $\leq r$-stars with center in $C_X^{j-1} \setminus C_X^j$).

the A'-star cover property is trivially fulfilled. The restricted X-neighborhood property, that is, that there is no edge between B' and $X \setminus A' = F_X$, is fulfilled, since all the neighbors of F_X in Y are in $F_Y = N_G[N_J(F_X)] \setminus X$ and therefore not in $B' = Y \setminus F_Y$. The restricted Y-neighborhood property, that is, that there is no edge between B' and $N(X \setminus A') \setminus X = N_J(F_X)$, is fulfilled, since all the neighbors of $N_J(F_X)$ are in F_Y and therefore not in B'. □

We mention in passing that for the case $F_X = X$ all vertices in B' have distance at least three to vertices in X, as all vertices of distance at most two from vertices in X are in F_Y. Since X is a bdd-d-set of G, the vertices in B' thus have degree at most d, and all their neighbors have degree at most d as well. This is the reason why the vertices in B' do not have to be considered for a solution even when A' is empty.

Next, we deal with the more involved case that FindExtremal returns in line 7. We need to define some notation in order to be able to refer to the variables in FindExtremal in some particular iteration. Let F_X^j and F_Y^j be the sets F_X and F_Y, respectively, in the j-th call of ComputePacking (line 4). Furthermore, let (P^j, C_X^j, C_Y^j) be the output of the j-th call of ComputePacking$(J - (F_X^j \cup F_Y^j), X \setminus F_X^j, Y \setminus F_Y^j)$. Since $F_X^j = X \setminus C_X^{j-1}$ (line 6) it holds that $C_X^{j-1} = X \setminus F_X^j$. See Figure 4.7 for an illustration.

The key for the proof that FindExtremal returns two vertex sets A' and B' satisfying the local optimality conditions is the following result that, if FindExtremal iterated sufficiently many times, then every minimum-cardinality solution contains C_X.

Lemma 4.9. *For $j \geq \lceil 1/\epsilon \rceil + 1$ and $d \geq 2$, the set C_X^j is contained in every minimum-cardinality bdd-d-set of G.*

The proof of Lemma 4.9 is given below. Before that, we use Lemma 4.9 to show that the two sets (A', B') returned by FindExtremal in line 7 fulfill the local optimality conditions.

Proposition 4.3. *If $j \geq \lceil 1/\epsilon \rceil + 1$ for $d \geq 2$, then the following properties hold for the vertex subsets (A', B') returned by* FindExtremal *in line 7:*

(1) If S' is a bdd-d-set of the induced subgraph $G-(A'\cup B')$, then $A'\cup S'$ is a bdd-d-set of G.
(2) There is a minimum-cardinality bdd-d-set S of G with $A'\subseteq S$.

Proof. We first show the second property. Since F_X^j is set to $X\setminus C_X^j$ in line 6, $A'=X\setminus F_X^j=C_X^j$, and hence by Lemma 4.9 there exists a minimum-cardinality bdd-d-set S such that $A'\subseteq S$.

Next, we show the first property. Let S' be a bdd-d-set of $G-(A'\cup B')$. To show that $A'\cup S'$ is a bdd-d-set of G, it suffices to show that all vertices in $N[B']\setminus A'$ have degree at most d in $G-(A'\cup S')$. Since X is a bdd-d-set of G, $X\setminus A'=F_X^j$ is a bdd-d-set for $G-A'$. Therefore, in $G-A'$, the only vertices that possibly have degree more than d are in F_X^j or adjacent to vertices in F_X^j. Since F_Y^j is set to $N_G[N_J(F_X^j)]\setminus X$ (line 6), neither the vertices in $B'=Y\setminus F_Y^j$ nor their neighbors $N(B')$ can be in F_X^j or $N_J(F_X^j)$, and thus the vertices in $N[B']\setminus A'$ have degree at most d in $G-A'$ and therefore also in $G-(A'\cup S')$. Hence, $A'\cup S'$ is a bdd-d-set of G, which shows the first property. □

With Proposition 4.2 and Proposition 4.3, Statement 4.2 follows immediately.

It remains to show Lemma 4.9. To this end, we first prove the following lemma.

Lemma 4.10. *Suppose that $d\geq 2$ and $C_X^j\neq X\setminus F_X^j$. If for a minimum-cardinality bdd-d-set S it holds that $|C_X^j\setminus S|=l$, then $|C_X^{j-1}\setminus S|\geq l\cdot\lceil|X|^\epsilon\rceil$.*

Proof. The proof approach is to assume that $|C_X^{j-1}\setminus S|<l\cdot\lceil|X|^\epsilon\rceil$ and to show that then there exists a bdd-d-set of G that is smaller than S, contradicting the assumption that S is a minimum-cardinality bdd-d-set.

First, for a simpler presentation of the main argument, assume that v is the only vertex in C_X^j that is not in S, that is, $C_X^j\setminus S=\{v\}$. Due to Lemma 4.7, each vertex of C_X^j is the center of an r-star in the star packing P^j. Since P^j is a star packing for $J-(F_X^j\cup F_Y^j)$, all the leaves of these stars are in $Y\setminus F_Y^j$. Recall that $r=d+1+\lceil|X|^\epsilon\rceil$ (cf. Section 4.3.4). Since $v\notin S$, at least $r-d=\lceil|X|^\epsilon\rceil+1$ leaves of the r-star in P^j with center v must be in S, since otherwise v would have degree more than d in $G-S$. Let S_v be these leaves. If $C_X^{j-1}\setminus S$ contains less than $\lceil|X|^\epsilon\rceil$ vertices, then one obtains a smaller bdd-d-set S' by setting $S':=(S\setminus S_v)\cup(C_X^{j-1}\setminus S)$, contradicting the assumption that S is minimum; the set S' is clearly smaller than S, and one can show that S' is a bdd-d-set as follows. We only have to verify that each vertex in $N[S_v]\setminus S'$ has degree at most d in $G-S'$. Clearly, $C_X^{j-1}=X\setminus F_X^j\subseteq S'$. Since X is a bdd-d-set of G, the only vertices in $G-S'$ that could have degree more than d are in F_X^j and $N_J(F_X^j)$. Since F_Y^j is set to $N_G[N_J(F_X^j)]\setminus X$ (line 6), neither the vertices in $S_v\subseteq Y\setminus F_Y^j$ nor their neighbors $N(S_v)$ can be in F_X^j or $N_J(F_X^j)$. Thus, the vertices in $N[S_v]\setminus S'$ have degree at most d in $G-S'$. This shows that S' is a bdd-d-set of G.

Thus, $C_X^{j-1}\setminus S$ contains at least $\lceil|X|^\epsilon\rceil$ vertices. One can show in complete analogy that if $C_X^j\setminus S$ contains l vertices, then $C_X^{j-1}\setminus S$ contains at least $l\cdot\lceil|X|^\epsilon\rceil$ vertices. □

Now, we are ready to prove Lemma 4.9, that is, that for $j \geq \lceil 1/\epsilon \rceil + 1$ and $d \geq 2$, the set C_X^j is contained in every minimum-cardinality bdd-d-set of G.

Proof of Lemma 4.9. In order to show the lemma, that is, that C_X^j is contained in every minimum-cardinality bdd-d-set, we assume that there exists a minimum-cardinality bdd-d-set S such that $C_X^j \not\subseteq S$, and show a contradiction by proving that S cannot have minimum-cardinality.

By assumption, $C_X^j \setminus S$ contains at least one vertex. By Lemma 4.10, then $C_X^{j-1} \setminus S$ contains at least $\lceil |X|^\epsilon \rceil$ vertices. By a repeated application of Lemma 4.10, we obtain that $C_X^1 \setminus S$ contains at least $\lceil |X|^\epsilon \rceil^{j-1} \geq |X|^{\lceil 1/\epsilon \rceil \cdot \epsilon} \geq |X|$ vertices. However, for each vertex in $C_X^1 \setminus S$ there is a vertex-disjoint r-star (Lemma 4.7), where $r = d + 1 + \lceil |X|^\epsilon \rceil$ (cf. Section 4.3.4), and hence S would have to contain more than $|X|$ vertices in order to be a bdd-d-set. This is a contradiction to the assumption that S has minimum cardinality, since X is a bdd-d-set of G and therefore $|X|$ is a trivial upper bound of the size of a minimum-cardinality bdd-d-set of G. This shows that every minimum-cardinality bdd-d-set S contains C_X^j. □

In summary, FindExtremal always returns two sets A' and B' satisfying the local optimality conditions. It remains to show Statement 4.3, that is, that the returned set B' is not empty. The key to showing this is to prove that there cannot be too many forbidden vertices in F_Y compared to F_X.

Number of Forbidden Vertices

Recall that one of the preconditions of Statement 4.3 is that Y contains more than $(d+1)^2 \cdot |X|$ vertices for $d \leq 1$ or more than $O(|X|^{1+\epsilon})$ vertices for $d \geq 2$. The point is, as we will show, that the set F_Y in FindExtremal always contains at most $(d+1)^2 \cdot |F_X|$ vertices for $d \leq 1$ or at most $O(|F_X|^{1+\epsilon})$ vertices for $d \geq 2$. Hence, the set $B' := Y \setminus F_Y$ returned by FindExtremal can never be empty.

Lemma 4.11. *For each $j \geq 1$, the set F_Y^j has size at most $r(1 + d + dj(d-1)^j) \cdot |F_X^j|$.*

Proof. We recall the definitions of some important notations. Let F_X^j and F_Y^j be the sets F_X and F_Y, respectively, in the j-th call of ComputePacking (line 4 of FindExtremal). Furthermore, let (P^j, C_X^j, C_Y^j) be the output of the j-th call of ComputePacking$(J - (F_X^j \cup F_Y^j), X \setminus F_X^j, Y \setminus F_Y^j)$. Note that $F_X^1 = F_Y^1 = \emptyset$ (line 2 of FindExtremal). Since $F_X^j = X \setminus C_X^{j-1}$ (line 6 of FindExtremal) it holds that $C_X^{j-1} = X \setminus F_X^j$. See Figure 4.8 for an illustration that shows the variables that are important in this proof. Recall that $F_Y^j = N_G[N_J(F_X^j)] \setminus X$ (line 6 of FindExtremal).

First, we bound the size of F_Y^2 with respect to F_X^2. Due to Lemma 4.7, $C_X^1 \cup C_Y^1$ is a vertex cover for J and thus there are no edges between $F_X^2 = X \setminus C_X^1$

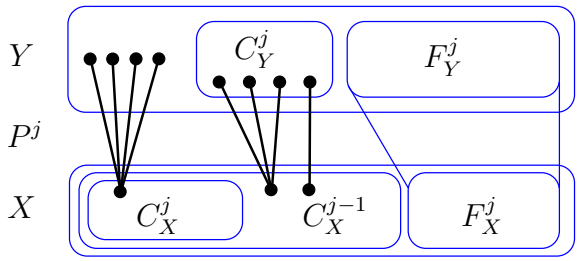


Figure 4.8: Illustration of the relation of the sets F_X^j, F_Y^j, C_X^j, C_Y^j, C_X^{j-1}, and the packing P^j for $r = 4$.

and $Y \setminus C_Y^1$. Hence, since due to Lemma 4.7 every vertex in C_Y^1 is the leaf of a $\leq r$-star with center in F_X^2, we have $N_J(F_X^2) = C_Y^1$, and $|N_J(F_X^2)| \leq r \cdot |F_X^2|$. Since X is a bdd-d-set of G, it follows that $|F_Y^2| = |N_G[N_J(F_X^2)] \setminus X| \leq r(d+1) \cdot |F_X^2|$.

Two important observations for the above size bound are that F_Y^2 contains all vertices that have distance at most one to a vertex in C_Y^1 in $G - X$, and that the vertices in C_Y^1 are leaves of $\leq r$-stars with center in F_X^2. We generalize these observations and show the general size bound of F_Y^j with respect to F_X^j. To this end, define

$$D_X^j := C_X^{j-1} \setminus C_X^j (\text{for } j \geq 2).$$

Hence, D_X^j is exactly the set $F_X^{j+1} \setminus F_X^j$ (informally speaking, the set of vertices that are added to F_X in the j-th iteration of FindExtremal), and $N_G[N_J(D_X^j)] \setminus X$ contains all vertices in $F_Y^{j+1} \setminus F_Y^j$.

By Lemma 4.7, $C_X^j \cup C_Y^j$ is a vertex cover of $J - (F_X^j \cup F_Y^j)$, and the vertices in C_Y^j are the leaves of $\leq r$-stars with center in D_X^j. Thus, there are no edges between D_X^j and $Y \setminus (F_Y^j \cup C_Y^j)$ and therefore $N_J(D_X^j)$ contains the vertices in C_Y^j and possibly also vertices in F_Y^j, but no vertices in $Y \setminus (F_Y^j \cup C_Y^j)$ (cf. Figure 4.8). The number of vertices in C_Y^j is easy to bound: since the vertices in C_Y^j are the leaves of a $\leq r$-star packing with centers in D_X^j, we have

$$|C_Y^j| \leq r \cdot |D_X^j|.$$

For a vertex v of $N_J(D_X^j) \cap F_Y^j$, observe that v is in F_Y^j because either it is in $C_Y^{j'}$ for some $j' < j$, or there is a path in $G - X$ of length at most $j - j'$ from v to a vertex in $C_Y^{j'}$ for some $j' < j$. Hence, for $1 \leq j' < j$, in j iterations the algorithm FindExtremal can only add vertices to F_Y^j that are at distance at most $j - j'$ from a vertex in $C_Y^{j'}$ in $G - X$. To simplify the analysis, for each $1 \leq j' < j$, we bound the number of vertices at distance at most $j + 1$ from $C_Y^{j'}$ in $G - X$. Since $G - X$ has bounded degree d, the number of all vertices at distance at

most $j+1$ from $C_Y^{j'}$ in $G-X$ (including the vertices in $C_Y^{j'}$) can be bounded by

$$\begin{aligned} & r \cdot |D_X^{j'}| + rd \cdot |D_X^{j'}| + rd(d-1) \cdot |D_X^{j'}| \\ & \quad + rd(d-1)^2 \cdot |D_X^{j'}| + \cdots + rd(d-1)^j \cdot |D_X^{j'}| \\ & \quad = |D_X^{j'}| \cdot r \left((1+d) + d((d-1) + (d-1)^2 + \cdots + (d-1)^j)\right) \\ & \quad \leq |D_X^{j'}| \cdot r \left((1+d) + dj(d-1)^j\right). \end{aligned}$$

In total, since $F_X^j = \bigcup_{1 \leq j' < j} D_X^{j'}$ and $D_X^{j'} \cap D_X^{j''} = \emptyset$ for $j' \neq j''$ (by the definition of D_X^j), we obtain

$$\begin{aligned} |F_Y^j| &\leq \sum_{1 \leq j' < j} |D_X^{j'}| \cdot r \left(1 + d + dj(d-1)^j\right) \\ &= |F_X^j| \cdot r \left(1 + d + dj(d-1)^j\right). \end{aligned}$$

□

With Lemma 4.11 one can now show the following proposition, which proves Statement 4.3.

Proposition 4.4. *Let $G = (V, E)$ be an undirected graph and let X be a bdd-d-set of G. If $Y = V \setminus X$ contains more than $(d+1)^2 \cdot |X|$ vertices for $d \leq 1$ or more than $c' \cdot |X|^{1+\epsilon}$ vertices for $d \geq 2$ (for some c' depending on d and ϵ), then algorithm* FindExtremal *in Figure 4.5 returns two vertex subsets (A', B') such that B' is not empty.*

Proof. For $d \leq 1$ (recall that $r = d+1$ in this case, cf. Section 4.3.4), if FindExtremal returns (A', B') in line 5, then by Lemma 4.11 one knows that $|F_Y| \leq |F_X| \cdot (d+1)^2$. Since Y contains more than $(d+1)^2 \cdot |X|$ vertices and since $F_X \subseteq X$, $B' = Y \setminus F_Y$ cannot be empty.

For $d \geq 2$ (recall that $r = d + 1 + \lceil |X|^\epsilon \rceil$ in this case), if FindExtremal returns (A', B') (in line 5 or line 7), then $j \leq \lceil 1/\epsilon \rceil + 1$ (condition in line 7) and since $F_X \subseteq X$ one knows by Lemma 4.11 that

$$\begin{aligned} |F_Y| &\leq (d + 1 + \lceil |X|^\epsilon \rceil) \left(1 + d + d(\lceil 1/\epsilon \rceil + 1)(d-1)^{\lceil 1/\epsilon \rceil + 1}\right) \cdot |X| \\ &\leq c' \cdot |X|^{1+\epsilon} \text{ (for some } c' \text{ depending on } d \text{ and } \epsilon). \end{aligned}$$

Hence, since Y contains more than $c' \cdot |X|^{1+\epsilon}$ vertices, $B' = Y \setminus F_Y$ cannot be empty. □

This finishes the proof of the local optimization theorem for Bounded-Degree-d Vertex Deletion (Theorem 4.2).

Remarks. Note that the entire local optimization algorithm is based on packing stars. For example, for $d = 1$, it is based on a packing of stars with two leaves (P_3). We can use our local optimization algorithm also for the problem of packing at least k copies of P_3 in a given graph G, called P_3-Packing. First, we compute again a packing of stars with two leaves. If we find at least k stars, then we abort returning "yes-instance". Otherwise, let the set X contain the vertices of the less than k stars, and proceed with the kernelization as described. To show that the algorithm returns two vertex subsets A' and B' satisfying the A'-star cover property for $d = 1$, we used the fact that there is a packing of 2-stars in $G[A' \cup B']$ such that each vertex in A' is the center of one such star (and the leaves of these 2-stars are therefore in B'). Moreover, the restricted neighborhood properties also imply that there is no P_3 using vertices from B' and $G \setminus A'$, thus using the stars in $G[A' \cup B']$ is always optimal. The size bound and the remaining proof then are exactly the same. Thus, we obtain the following result.

Corollary 4.2. *P_3-Packing admits a problem kernel of $15k$ vertices.*

The currently best problem kernel for P_3-Packing has $7k$ vertices [WNFC08]. This improvement stems basically from some local modification of an initial maximal P_3-packing and would also work with our technique.

The main point we want to make here is that there seems to be a close relationship between the kernelizations for star packing problems and Bounded-Degree-d Vertex Deletion, and similar observations also hold for other packing/deletion problem pairs (cf. Chapter 7). Note that the problem of packing at least k stars of more than two leaves ($K_{1,l}$-Packing for constant l) admits a problem kernel of $O(k^2)$ vertices [PS06]. It is conceivable that our technique also works for this problem. However, to this end, one would have to provide a new proof of Proposition 4.3.

4.4 Search Tree Algorithm

In this section, we describe two simple bounded search tree algorithms to solve Bounded-Degree-d Vertex Deletion. We give an algorithm for general constant d and a faster algorithm for $d = 1$. The worst-case exponential part of the running time of these two algorithms is not better than in the running time resulting by reduction to Hitting Set or in the running time of an algorithm that enumerates all minimal solutions [KHMN09] (see Section 4.1). However, the algorithms presented in this section can be interleaved with kernelization algorithms that are particularly tailored for Bounded-Degree-d Vertex Deletion (see Chapter 9), but which are not suitable for enumeration algorithms.

4.4.1 General Branching Strategy

The first search tree algorithm solves the Bounded-Degree-d Vertex Deletion problem for any constant d.

Theorem 4.3. Bounded-Degree-d Vertex Deletion *can be solved in* $O((d+2)^k + n(k+d))$ *time.*

Proof. Let (G, k) be an instance of Bounded-Degree-d Vertex Deletion. While the graph has not bounded degree d, we choose an arbitrary vertex $v \in V(G)$ with $\deg(v) > d$. Clearly, v either needs to be deleted or all but d of its neighbors to achieve maximum degree d. Therefore, we branch into the following cases:

1. If v is part of the solution, then delete v from G and decrease k by one.
2. If v is no part of the solution, then choose an arbitrary subset of $d+1$ neighbors of v, and, for each vertex w of the subset, branch into a subcase by deleting w from G and decreasing k by one.

In each subcase of the branching we put at least one vertex from G into the solution (by deleting it), and we branch into $d+2$ subcases. This results in a search tree of size $O((d+2)^k)$. Testing whether a graph has bounded degree d can be done in $O(dn)$ time. In each search tree node, we kernelize the instance before branching, using the simple $O(k^2)$-vertex problem kernel (Theorem 4.1). We chose this problem kernel due to its faster running time $O(n(k+d))$ compared to the kernel based on the local optimization theorem (Corollary 4.1). In total, this yields a running time of $O((d+2)^k \cdot k^2 + n(k+d))$. An improved *interleaving* of the depth-bounded search tree with a problem kernel, which only kernelizes the instance in a search tree node if the instance is larger than a certain threshold [NR00], yields a running of $O((d+2)^k + n(k+d))$. □

Note that a similar search tree algorithm is used to solve Regular-Degree Vertex Deletion in Section 5.3.

4.4.2 Improved Branching Strategy for $d = 1$

For the practically relevant special case $d = 1$ (cf. Chapter 9), we give a more refined branching strategy with an improved search tree size of $O(2.31^k)$. We refrain from conceivable further asymptotic improvements (which appear likely when using even further refined branching strategies) in order to keep the algorithm easy to implement and efficient by avoiding the overhead incurred by more complicated strategies.

Theorem 4.4. Bounded-Degree-1 Vertex Deletion *can be solved in* $O(2.31^k + kn)$ *time.*

Proof. We start with considering a vertex v of degree $t > 1$. Clearly, v either needs to be deleted or all but one of its neighbors to achieve maximum degree one. Let $N(v) = \{u_1, \ldots, u_t\}$. We branch into the following $t + 2$ subcases:

1. Delete v from G and decrease k by one.
2. Delete $N(v)$ from G and decrease k by $|N(v)|$.
3. For each $u_i \in N(v)$, $1 \leq i \leq t$, delete $N' := (N(v) \setminus \{u_i\}) \cup (N(u_i) \setminus \{v\})$ from G and decrease k by $|N'|$.

The correctness of this branching can be seen as follows. First, clearly in each subcase v either gets deleted (case 1) or it gets maximum degree one (degree zero in case 2 and degree one in case 3). Second, the branching covers all possibilities how v can be made a maximum-degree-one vertex: one can keep at most one vertex from $N(v)$ (case 3), the rest has to be deleted. If u_i is the neighbor that shall not be deleted, then clearly all vertices from $N(v) \setminus \{u_i\}$ have to be deleted (otherwise, v would have degree greater than one) and all neighbors of u_i except for v (that is, $(N(u_i) \setminus \{v\})$ have to be deleted (otherwise, u_i would have degree greater than one). The case of deleting all of $N(v)$ (case 2) also needs to be considered since, otherwise, one would miss the situation that all of v's neighbors have to be deleted for reasons lying outside the neighborhood of v (in other words, if we would not consider case 2, we would miss the case that v obtains degree zero). One obtains a branching into $t+2$ cases with the corresponding branching vector

$$(1, t, t - 1 + |N(u_1) \setminus N[v]|, \ldots, t - 1 + |N(u_t) \setminus N[v]|).$$

It is not hard to check[2] that the worst-case branching vector occurs for $t = 2$ and $|N(u_1) \setminus N[v]| = |N(u_2) \setminus N[v]| = 1$, yielding $(1, 2, 2, 2)$ with the branching number 2.31. In analogy to the search tree algorithm for Bounded-Degree-d Vertex Deletion for general d (Section 4.4.1) we can interleave the search tree with problem kernelization, which results in $O(2.31^k + kn)$ running time in total. □

We have implemented and experimentally tested these branching strategies (Theorem 4.3 and Theorem 4.4), interleaving them with several kernelization methods. See Chapter 9 for more details.

4.5 Iterative Compression for $d = 1$

In this section, we give an improved algorithm for Bounded-Degree-1 Vertex Deletion using the iterative compression technique (see Section 2.3.3). We

[2]We omit some details here; basically, one can argue that for $t = 2$ cases where $|N(u_1) \setminus N[v]| = 0$ are actually easier (often avoiding branching at all) and $t > 2$ gives branching vectors with smaller branching numbers.

remark that the overall strategy used in the following is very similar to the one known for Cluster Vertex Deletion as described in Section 6.3.1. The fundamental difference, in terms of corresponding forbidden subgraph characterizations, is that in the case of Cluster Vertex Deletion the task is to destroy all *induced* P_3's (paths of three vertices) whereas in the case of Bounded-Degree-1 Vertex Deletion the task is to destroy all P_3's. We will see that this permits a faster compression routine compared to Cluster Vertex Deletion.

We first describe how to employ an assumed compression routine, and then the compression routine itself. The general idea behind our iterative compression is as follows. Start with $V' = \emptyset$ and $S = \emptyset$; clearly, S is a bdd-1-set for $G[V']$. Iterating over all graph vertices, step by step add one vertex $v \notin V'$ from V to both V' and S. Then, S is still a bdd-1-set for $G[V']$, although possibly not a minimum one. One can, however, obtain a minimum one by applying a compression routine. It takes a graph G and a bdd-1-set S for G, and returns a minimum-cardinality bdd-1-set for G. Since eventually $V' = V$, one obtains an optimal solution for G once the algorithm returns S. Hence, the main task is to design a compression routine. Consider a smaller bdd-1-set S' as the modification of the larger bdd-1-set S for the graph $G = (V, E)$. This modification retains some vertices $Y \subseteq S$ as part of the solution set, while the other vertices in $X := S \setminus Y$ are replaced by new vertices X' from $V \setminus S$, where $|X'| < |X|$. The idea is to try by brute force all $2^{|S|} - 1$ nontrivial partitions of S into these two sets Y and X. For each such partition, the vertices from Y are immediately deleted, since we already decided to put them into the smaller bdd-1-set. Now, the task is either to find a bdd-1-set X' of size less than $|X|$ for the remaining graph or to prove that no such X' exists. We call this remaining task Disjoint Compression Task. Clearly, one requirement for the existence of a disjoint bdd-1-set X' is that $G[X]$ has maximum degree one (that is, it is P_3-free). This requirement can be verified in linear time, therefore, in the following we assume that it is satisfied.

For the compression routine, let $G = (V, E)$ and $X \subseteq V$ with $G - X$ being a graph without a P_3 as a subgraph, that is, $G - X$ has maximum degree one. Let $R := V \setminus X$. As explained above, we can also assume that $G[X]$ has maximum degree one. First, compute a set $X_1 \subseteq R$ of vertices that necessarily belong to the disjoint solution by applying a series of polynomial-time executable data reduction rules. Then, compute a set $X_2 \subseteq (R \setminus X_1)$ such that $X' := X_1 \cup X_2$ forms a minimum-size solution. If in this process one encounters a situation that shows that the given instance has no solution, then return "no-instance". We start with the description of the data reduction rules. Initially, set $X_1 := \emptyset$. The four data reduction rules read as follows.

1. For each edge $\{u, v\}$ in $G[X]$, set $X_1 := X_1 \cup (R \cap (N(u) \cup N(v)))$, delete u and v from G and X, and delete $R \cap (N(u) \cup N(v))$ from G and R.

2. Delete each vertex $u \in R$ that is adjacent to more than one vertex in X from G and R and set $X_1 := X_1 \cup \{u\}$.

3. For each edge $\{u, v\}$ in $G[R]$ such that $|N(\{u, v\})| = 1$, choose a vertex $x \in \{u, v\}$ that is adjacent to the vertex in $N(\{u, v\})$, set $X_1 := X_1 \cup \{x\}$ and delete x from G and R.

4. Delete isolated vertices and isolated edges in R from G and R.

The correctness of the first two reduction rules is due to the fact that each P_3 that intersects with X in two vertices can only be destroyed if the third vertex (from R) is in X_1. The third reduction rule is correct, because the only neighbor of $\{u, v\}$ is the vertex in $N(\{u, v\}) = \{w\}$, thus $\{u, v, w\}$ induces either a triangle (K_3) or a path on three vertices P_3. As a consequence, it is optimal to add a vertex from $\{u, v\}$ that is adjacent to w to X_1. The fourth reduction rule is obviously correct, as an optimal solution would never contain isolated vertices or a vertex in an isolated edge.

These reduction rules can be easily applied exhaustively in $O(n + m)$ time if they are applied in the given order: To apply the first reduction rule, iterate through each edge in $G[X]$ and delete its neighbors in R, which takes $O(n + m)$ time in total. Then, for the second rule, simply check for each vertex u in R if there are at least two neighbors in X, and if so, delete u, which takes $O(n + m)$ for all vertices in R. To apply the third rule, iterate over each edge $\{u, v\}$ in $G[R]$ and test whether the neighborhood of $\{u, v\}$ contains only one vertex, in which case either u or v is deleted. In total, this takes $O(n + m)$ time. The fourth rule clearly takes $O(n + m)$ time. It is important to observe that after one particular reduction rule has been exhaustively applied, there can never again occur a situation where the same rule could be applied again. This guarantees that after applying the rules in their given order (the order is important, e.g., applying the fourth rule before the others could lead to an instance that contains isolated vertices or edges), the instance is reduced with respect to the four data reduction rules. Hence, an instance can be exhaustively reduced in $O(n + m)$ time.

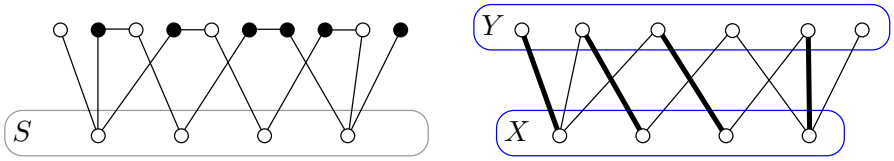
After these rules have been exhaustively applied, if $|X_1| > |X|$, then stop and return "no-instance". Otherwise, compute a minimum-size set X_2 such that $X_1 \cup X_2$ is a minimum bdd-1-set as follows.

In the following, assume that G, X, and R are reduced with respect to the data reduction rules, that is, that none of the above rules can be applied anymore. The following properties of G, X, and R are important for the subsequent arguments.

P1 The graph $G[R]$ consists of isolated vertices and edges (because X obstructs all P_3's),

P2 X forms an independent set (first reduction rule),

P3 each vertex in R is adjacent to exactly one vertex in X (reduction rules two to four), and



(a) Graph G with vertex set X, preprocessed by the four data reduction rules.

(b) Corresponding Maximum Matching instance.

Figure 4.9: Reduced Disjoint Compression Task instance (a) together with a corresponding Maximum Matching instance (b). The bold edges in the Maximum Matching instance form a maximum matching M. The black vertices in G are the vertices of a minimum-size bdd-1-set X_2, $X_2 \cap X = \emptyset$, corresponding to M.

P4 each vertex in X is adjacent to at most one endpoint of each edge in $G[R]$ (P3 and third reduction rule).

An example of an instance with these properties is given in Figure 4.9a. Observe that for each edge in $G[R]$ at least one of its endpoints must belong to the new solution X' in order to obstruct all P_3's. Moreover, for each vertex $v \in X$, all but at most one neighbor must belong to X'. An optimal solution that fulfills these constraints can be easily found by reduction to Maximum Matching; the corresponding Maximum Matching instance is constructed by contracting every edge in R, and a maximum matching M in that instance directly corresponds to a solution X_2 in G (see Figure 4.9b for an example). Obviously, the input instance (G, k) is a yes-instance for Bounded-Degree-1 Vertex Deletion if and only if $|X_1| + |X_2| \leq |X|$.

It remains to show the running time to solve the matching instance. Observe that in the Maximum Matching instance the vertices in one partite set have maximum degree two. Such an instance can be solved in linear time:

Lemma 4.12. *On a bipartite graph B with partite vertex sets X and Y such that the vertices in Y have maximum degree two,* Maximum Matching *can be solved in $O(n + m)$ time, where n and m are the number of vertices and edges of B, respectively.*

Proof. The following is a description of an algorithm that finds a maximum matching M in B. It starts with an empty set M and then adds edges to it as follows.

First of all, one can safely assume that an edge incident to a degree-one vertex is always contained in a maximum matching; if an edge $\{u, v\}$, where u has degree one, is not in a maximum matching M, then there must exist an edge $\{v, w\} \in M$ for some vertex w and we can simply remove $\{v, w\}$ from M and replace it by $\{u, v\}$.

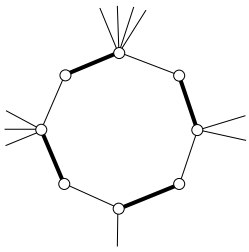


Figure 4.10: A cycle C with vertices of degree two ("inner vertices") and vertices of degree at least two ("exits"). Any maximum matching M can contain at most four exits, and any matching edge that contains an inner vertex also contains an exit. Therefore, taking every second edge on C into the matching is always optimal. The bold edges show such a matching on C.

Therefore, in a first step, we find in $O(n)$ time all degree-one vertices. Then, repeat the following until there are no more degree-one vertices. Choose a degree-one vertex u, add its incident edge $\{u, v\}$ to M, and remove v from B. Note that removing v causes that its degree-one neighbors obtain degree zero and it might cause that some of its neighbors obtain degree one. Thus, the set of degree-one neighbors is updated accordingly. The whole process of removing all degree-one vertices takes $O(n + m)$ time; there are at most n vertices to be removed, and it takes $O(\deg(v))$ time to update the set of degree-one neighbors if v is removed, which sums up to $O(m)$ in total.

The remaining graph has minimum degree two. Hence, each connected component contains a cycle. The connected components and a cycle in each of it can be found in $O(n + m)$ time with a depth-first search.

From now on, let B' be a connected component with partite vertex sets $X' \subseteq X$ and $Y' \subseteq Y$ and let C be a cycle in B'. Obviously, C has even length because B' is bipartite. Since the graph has minimum degree two, the vertices in Y' have degree exactly two, and therefore every second vertex on C has degree exactly two. Moreover, every vertex in $D := N(V(C))$ is from Y' and has degree two. There exists a maximum matching M that contains every second edge on C; any maximum matching M' without that property can be easily converted to a maximum matching of the same size with that property (see Figure 4.10). Thus, we add every second edge on C to M and remove $V(C)$ from B'. After that, the neighbors D of the removed cycle have degree one, and therefore we can apply the same process of removing all degree-one vertices as described above. This process will remove the whole connected component B' for the following reason. Let $v \in D$ and let w be its neighbor (we assume that C has been removed). Since $v \in Y'$, it follows that $w \in X'$, and since the process of removing degree-one vertices will remove w, the degree-two neighbors of w will obtain degree one. By applying this argument inductively, it is clear that the process of removing degree-one vertices will remove every vertex w' for which there exists a path between v

and w' in B'. Thus, eventually, all vertices in the connected component B' are removed. Clearly, deleting $V(C)$ takes linear time, and, as explained above, the subsequent removal of degree-one vertices also takes linear time.

This shows that we can find a maximum matching in B in $O(n+m)$ time. □

Using this linear-time algorithm to solve the Maximum Matching instance, the Disjoint Compression Task for Bounded-Degree-1 Vertex Deletion can be solved in $O(n + m)$ time. Now, we can finish the analysis of the whole iterative compression algorithm.

Theorem 4.5. Bounded-Degree-1 Vertex Deletion *can be solved in* $O(2^k \cdot k^2 + kn)$ *time.*

Proof. Using Theorem 4.1, we get an $O(k^2)$-vertex problem kernel for Bounded-Degree-1 Vertex Deletion (with $O(k^4)$ edges) in $O(kn)$ time. On this kernel, we apply Theorem 4.2 and obtain an $O(k)$-vertex problem kernel for Bounded-Degree-1 Vertex Deletion (with $O(k^2)$ edges) in $O(k^{12})$ time. We apply the above iterative compression algorithm on that problem kernel. This means that we have $O(k)$ iterations, each taking $O(2^k \cdot k^2)$ time. Herein, the factor $O(2^k)$ derives from trying all partitions of X into two subsets. The resulting total running time is $O(kn + k^{12} + 2^k \cdot k^2) = O(2^k \cdot k^2 + kn)$. □

Note that Bounded-Degree-1 Vertex Deletion is exceptional in the sense that the corresponding Disjoint Compression Task can be solved in linear time. For other related problems (like Cluster Vertex Deletion), whose corresponding Disjoint Compression Task are polynomial-time solvable, it seems to be much more difficult to reach linear-time solvability (cf. Section 6.3.1).

4.6 Hardness

Our results in Section 4.3 show that Bounded-Degree-d Vertex Deletion is fixed-parameter tractable with respect to the parameter k if d is a constant. However, as we will prove in this section, the problem becomes presumably fixed-parameter intractable for unbounded d—in other words, we show it to be W[2]-complete.

Preliminaries. A parameterized problem L is contained in W[2] if there is a parameterized reduction from L to the Weighted Satisfiability problem for polynomial-size weft-two circuits of constant depth [DF98]. Herein, the *weft* of a circuit C is the maximum number of "large" gates on an input-output path in C. In a Boolean circuit, a gate ($\neg$, $\wedge$, $\vee$) is *small* if it has fan-in bounded by a function of the parameter k, whereas *large* gates have unbounded fan-in. The *depth* of a circuit C is defined as the maximum number of gates on an input-output path in C. The *weight* of a truth assignment is the number of variables that are set

true. To show W[2]-hardness, we employ the W[2]-complete Dominating Set problem (see, e.g., [DF99]).

Dominating Set
Input: An undirected graph $G = (V, E)$ and an integer $k \geq 0$.
Question: Does there exist a vertex subset $S \subseteq V$ of size at most k such that every vertex of V belongs to S or has a neighbor in S?

Theorem 4.6. *For d being unbounded,* Bounded-Degree-d Vertex Deletion *is W[2]-complete with respect to the parameter k.*

Proof. The W[2]-hardness of Bounded-Degree-d Vertex Deletion can be easily shown by a parameterized reduction from the W[2]-complete Dominating Set problem: Pad the vertices in the Dominating Set instance with degree-one neighbors such that every vertex has the same degree. Let $d + 1$ be the degree of the resulting regular graph. For each original vertex, at least one neighbor or the vertex itself has to be removed in order to obtain maximum degree d (we assume without loss of generality that no newly added degree-one vertex is removed by an optimal solution), which directly corresponds to a dominating set in the Dominating Set instance.

Second, we show the membership of Bounded-Degree-d Vertex Deletion in W[2]. Let $(G = (V, E), k, d)$ be an instance of Bounded-Degree-d Vertex Deletion. We construct a Boolean circuit of weft two and constant depth, where small gates have fan-in bounded by an arbitrary fixed function of k. This shows membership in W[2].

The Boolean circuit is given by a Boolean expression E that is satisfiable by a weight-k truth assignment if and only if G has a k-vertex solution to Bounded-Degree-d Vertex Deletion.

The informal idea of the construction is as follows: We have k choices to select vertices in V to be in the solution S. For each choice, we introduce a *block* of $|V|$ Boolean variables. A Boolean subexpression E_1 will ensure that only one Boolean variable of each block can be set to true; the variable set to true in a block corresponds directly to the choice of the corresponding vertex to be in S. To avoid that a single vertex appears twice in a solution, we introduce a second subexpression E_2. Furthermore, we need Boolean subexpressions to express that a vertex $v \in V$ is in S (subexpression $E_3(v)$) or has to have at least $\deg(v) - d$ neighbors in S (subexpression $E_4(v)$). The complete Boolean expression can then be written as

$$E := E_1 \wedge E_2 \wedge \bigwedge_{v \in V^+} (E_3(v) \vee E_4(v)),$$

where V^+ is the set of vertices in V with degree at least $d+1$. Next, we formally describe the corresponding Boolean expressions:

The set of Boolean variables for E is

$$X := \{c[i, u] : 1 \leq i \leq k, u \in V\},$$

where $c[i,u]$ means that the ith choice of a vertex of S is vertex u. We define

$$E_1 := \bigwedge_{i=1}^{k} \bigwedge_{u,u' \in V, u \neq u'} \neg(c[i,u] \wedge c[i,u']),$$

meaning that no two variables in the same block can be set true,

$$E_2 := \bigwedge_{u \in V} \bigwedge_{1 \leq i < j \leq k} \neg(c[i,u] \wedge c[j,u]),$$

meaning that no two variables corresponding to the same vertex are set true, and

$$E_3(v) := \bigvee_{1 \leq i \leq k} c[i,v],$$

meaning that at least one variable corresponding to vertex v is set true in some block. Let $R(k,r)$ denote the set of size-r subsets of $\{1,\ldots,k\}$. Finally, we define

$$E_4(v) := \bigvee_{R' \in R(k,\deg(v)-d)} \bigwedge_{i \in R'} \bigvee_{u \in N(v)} c[i,u].$$

Informally speaking, this subexpression examines, for a given vertex v, every possible subset of blocks that is large enough to witness that sufficiently many neighbors (that is, at least $\deg(v) - d$) of v are chosen to be in the solution. Subexpression $E_4(v)$ checks for every block B in each such subset whether at least one variable of a neighbor of v is set true in B and returns true if this is the case for all blocks in the subset. Due to expression E_1 we know that then there are at least $\deg(v) - d$ neighbors of v chosen to be in the solution S.

One can easily verify that E is satisfiable by a weight-k truth assignment if and only if G has a k-vertex solution to Bounded-Degree-d Vertex Deletion. Moreover, the depth of the circuit is constant and the weft is two, as the only large gates (that is, with fan-in that is not bounded by a function of k) correspond to the outermost conjunction of E (over all $v \in V^+$), the inner conjunction $\bigwedge$ of E_1, the outermost conjunction of E_2, and the innermost disjunction of $E_4(v)$. All other gates have fan-in bounded by some function of k. □

4.7 Outlook

In this chapter, we have shown that there exists an almost linear-vertex problem kernel for Bounded-Degree-d Vertex Deletion with respect to the parameter k for any fixed d, that is, a kernel of $O(k^{\epsilon+1})$ vertices for any fixed d. For $d \leq 1$, the same method even gives a linear-vertex problem kernel. Moreover, we gave some simple kernelization that yields an $O(k^2)$-vertex kernel, and simple search tree algorithms. For $d = 1$, using the iterative compression technique, we gave the so far fastest algorithm for Bounded-Degree-1 Vertex

Deletion. Finally, we showed that Bounded-Degree-d Vertex Deletion becomes W[2]-complete with respect to the parameter k if d is unbounded; it is therefore presumably not fixed-parameter tractable.

There remain several open questions. First of all, does there exist a linear-vertex problem kernel for Bounded-Degree-d Vertex Deletion for any constant d? Moreover, for all known fixed-parameter algorithms for that problem, the exponential running time part is never better than $(d+1)^k$; a similar observation is true for Hitting Set, that is, all known algorithms for s-Hitting Set do not run better than $(s-1)^k$ (recall that Bounded-Degree-d Vertex Deletion can be solved via reduction to $(d+2)$-Hitting Set). However, Bounded-Degree-d Vertex Deletion is a specialization of Hitting Set, thus it would be interesting to see whether Bounded-Degree-d Vertex Deletion can be solved faster than the corresponding $(d+2)$-Hitting Set instance.

Chapter 5

Regular-Degree Vertex Deletion

In this chapter, we consider the problem of deleting at most k vertices from a given graph in order to obtain a *d-regular* graph for any fixed d. Compared to Bounded-Degree-d Vertex Deletion (see Chapter 4), where the underlying graph property is "each vertex has degree at most d", which is a hereditary graph property, here the underlying graph property is "each vertex has degree *exactly* d", which is a non-hereditary graph property. For almost all vertex deletion problems that have been studied in the parameterized context, the underlying graph property is hereditary; Regular-Degree-d Vertex Deletion is one of the few exceptions. The standard approaches to show NP-completeness and fixed-parameter (in-)tractability for vertex deletion problems cannot be applied here, because they rely on the fact that the underlying graph property is hereditary. We show that Regular-Degree-d Vertex Deletion is NP-complete even for planar graphs (for $d \leq 5$, since every planar graph contains a vertex of degree at most five) and triangle-free planar graphs (for $d \leq 3$). Moreover, it admits a problem kernel of $O(kd(d+k)^2)$ vertices, and it can be solved with a search tree algorithm in $O(n(k+d) + (d+2)^k)$ time. Thus, Regular-Degree-d Vertex Deletion is fixed-parameter tractable with respect to parameter k for constant d. We also prove that the dual problem, d-Regular Induced Subgraph, is W[1]-hard with respect to parameter k for any fixed d.

First, we show the problem kernel (Section 5.2). Then, we present the search tree algorithm (Section 5.3) and the hardness results (Section 5.4).

5.1 Introduction and Known Results

We consider the following dual parameterizations of the problem.

> Regular-Degree-d Vertex Deletion
> **Input:** An undirected graph $G = (V, E)$ and an integer $k \geq 0$.
> **Question:** Does there exist a vertex subset $S \subseteq V$ of size at most k such that $G - S$ is d-regular?

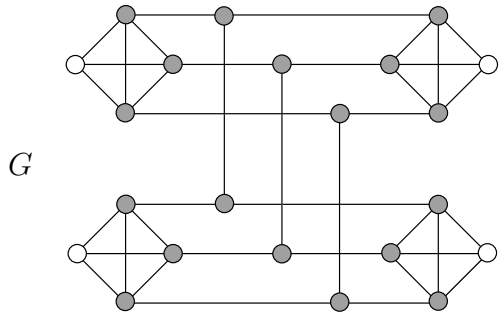


Figure 5.1: For 3-Regular Induced Subgraph, the graph G is a yes-instance if and only if $k \in \{0, 1, \ldots, 17, 18\}$. However, for its "exact version", the same graph is a yes-instance if and only if $k \in \{4, 8, 12, 16, 18\}$. For example, for $k = 18$, a 3-regular induced subgraph is marked with gray vertices.

d-Regular Induced Subgraph
Input: An undirected graph $G = (V, E)$ and an integer $k \geq 0$.
Question: Does there exist a vertex subset $S \subseteq V$ of size at least k such that $G[S]$ is d-regular?

Obviously, for $d = 0$, these problems are equivalent to Vertex Cover and Independent Set, respectively. The graph property "d-regular" is not hereditary, as an induced subgraph of a d-regular graph is not d-regular in general. Therefore, it makes a difference to ask for a solution set of size *exactly* k instead of at most k (Regular-Degree-d Vertex Deletion) or instead of at least k (d-Regular Induced Subgraph). See Figure 5.1 for an example. As the "non-exact" versions of the problems are weaker than (that is, can be reduced to) their "exact" counterparts, we prove all of our hardness results for the "non-exact" versions. However, in order to obtain a stronger statement, all algorithms in this chapter are designed for the "exact" versions of our problems (modifications for solving the "non-exact" versions are easy and have the same running times; note that it seems difficult to obtain algorithms for the "non-exact" version that are faster than algorithms for the "exact" version).

Because d-regularity is not a hereditary property, the framework of Yannakakis [LY80] can not be used to show NP-completeness, and neither can the framework by Khot and Raman [KR02] nor the algorithm by Cai [Cai96] be used to derive the parameterized complexity of Regular-Degree-d Vertex Deletion.

Applications. The 1-Regular Induced Subgraph problem is equivalent to Induced Matching, which has applications in (wireless) communication networks, secure communication channels, VLSI, and network flow problems. See Chapter 8 for more about Induced Matching and its applications. Moreover, regular induced subgraph problems find applications in game theory [BIL08]. There is also some relation to statistical physics [PW06].

Table 5.1: Parameterized complexity results for Regular-Degree Vertex Deletion (RDVD) and d-Regular Induced Subgraph (RIS) that are presented in this chapter.

problem	problem kernel	algorithm
RDVD	$O(kd(d+k)^2)$ (Sec. 5.2)	$O(n(k+d)+(d+2)^k)$ (Sec. 5.3)
RIS	W[1]-hard (Sec. 5.4)	

Known Results. The NP-completeness of Regular-Degree Vertex Deletion has also been shown independently by Cardoso et al. [CKL07]. Gupta et al. [GRS06] gave an algorithm for the maximization version of d-Regular Induced Subgraph with running time $O(c^n)$, where c is a positive constant strictly less than 2 and depending only on d. Moreover, they gave lower bounds on the number of maximal d-regular induced subgraphs. Cardoso et al. [CKL07] gave convex quadratic programs that derive an upper bound on the size of a d-regular induced subgraph of a given graph G, and they also studied such bounds assuming that G is regular. Cardoso and Pinheiro [CP09] gave further upper bounds based on convex quadratic programs and also tested their methods with computational experiments on graphs from the Second DIMACS clique challenge [DIM95].

The problem kernel in Section 5.2 and the algorithm in Section 5.3 (see Table 5.1 for an overview) have been a starting point for further research concerning the parameterized complexity of Regular-Degree-d Vertex Deletion: Stewart [Ste08] showed that the fixed-parameter tractability of Regular-Degree-d Vertex Deletion with respect to parameter k can be derived by means of general logical methods, without using the more direct approaches that are presented in this chapter. However, his method, based on a formulation of Regular-Degree-d Vertex Deletion in first-order logic, is only meant to establish fixed-parameter tractability, while our methods are more efficient. Mathieson and Szeider [MS08a] showed that (using a quite involved reduction from Multicolored Clique) Regular-Degree-d Vertex Deletion is W[1]-hard with respect to the parameter k if d is part of the input, which is equivalent to the problem of deleting at most k vertices to obtain a regular graph without specifying the degree in advance. Additionally to deleting vertices, they also considered edge deletion as allowed graph modification. Moreover, they introduced a weighted variant of the problem, where the desired vertex degree is specified for each vertex individually. For all their variants, they gave analogous kernelization and hardness results, where the kernelization results combine ideas from our kernelization in Section 5.2 and other ideas (e.g., annotation). In an additional paper, Mathieson and Szeider [MS08b] extended their work by adding edge insertion to the set of allowed graph modifications.

Related Problems. Regular graphs as well as regular subgraphs have been intensively studied from a structural point of view (e.g., [Big94]), and have also

a close relation to graph factors and factorization (see, e.g., the survey by Plummer [Plu07]). An interesting combinatorial problem related to regular graphs is to decide whether a given graph contains a d-regular subgraph. This problem is polynomial-time solvable for $d \leq 2$ [CM87]. Chvátal et al. [CFST79] showed that it is NP-complete for $d = 3$ (then also called Cubic Subgraph). In a series of papers, Stewart showed that Cubic Subgraph is NP-complete on planar graphs with maximum degree seven [Ste94] and bipartite graphs with maximum degree four [Ste97], and that for any fixed degree d the problem is NP-complete on general graphs as well as on planar graphs [Ste96] (where for the planar graphs only $d = 4$ and $d = 5$ were considered, since any planar graph contains a vertex of degree at most 5). Moreover, Cheah and Corneil [CC90] showed that it is NP-complete to decide whether a graph of maximum degree $d+1$ has a d-regular subgraph.

A problem that is more closely related to Regular-Degree-d Vertex Deletion is the problem of editing a given graph with at most k editing operations into a rectangular grid ("grid cleaning"), where the allowed edit operations are edge/vertex deletions and insertions. Díaz and Thilikos [DT06] showed that this problem, whose corresponding graph property is also non-hereditary, is fixed-parameter tractable with respect to the parameter k. Another "cleaning problem" is the Induced Subgraph Isomorphism problem, where the task is, given two graphs H and G, to find a set $S \in V(G)$ such that $G - S$ is isomorphic to H. Marx and Schlotter [MS09a] showed that this problem is fixed-parameter tractable with respect to the parameter $|S|$ if H is a tree and G an arbitrary graph, or if H is a 3-connected planar graph and G is a planar graph. If H is d-regular for any constant d, then there exists an exact algorithm solving Induced Subgraph Isomorphism in $O(c^n)$ time, where c is a constant strictly less than 2 and only depending on d [GRS06]. Bonifaci et al. [BIL08] showed that the problem of deciding the existence of a uniform Nash equilibrium in "imitation simple bimatrix games" is NP-complete by proving the NP-completeness of a related regular induced subgraph problem on directed graphs with the following "regularity condition": the in-degree of every vertex of the graph induced by the solution vertex set S is exactly d and all vertices outside of S have at most d in-neighbors in S. Bonifaci et al. [BIL08] also considered undirected graphs and showed that d-Regular Induced Subgraph, where the degree of the desired regular induced subgraph is not specified, is NP-complete. Moreover, they showed that, given an undirected graph and an integer k, to decide whether there exists a vertex subset S such that $G[S]$ is d-regular, for some $d \geq k$, is NP-complete.

5.2 A Cubic-Vertex Problem Kernel

The main result of this section is the following.

Theorem 5.1. Regular-Degree-d Vertex Deletion *admits a problem ker-*

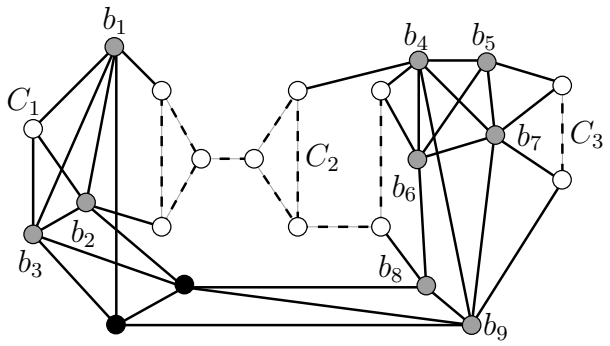


Figure 5.2: Example of a graph with clean regions C_1, C_2, and C_3 (white vertices, $d = 3$). The dotted edges denote the connected subgraph each clean region induces. Dirty vertices are gray or black; boundary vertices are gray and all other dirty vertices are black. The boundary for C_1 is $B_1 = \{b_1, b_2, b_3\}$, the boundary for C_2 is $B_2 = \{b_1, b_2, b_4, b_6, b_8\}$, and the boundary for C_3 is $B_3 = \{b_5, b_7, b_9\}$. Note that boundaries can have vertices in common, for instance, $B_1 \cap B_2 \neq \emptyset$.

nel of $O(kd(k+d)^2)$ vertices for $d \geq 2$ and of $O(k^2)$ vertices for $d = 1$, which can be constructed in $O(n(k+d))$ time.

The remainder of this section is dedicated to the proof of Theorem 5.1. Recall that we derive a kernel for the "exact" version of the problem, that is, we demand a solution of size exactly k (see Section 5.1). A kernel for the "non-exact" version can be easily derived by a minor modification; however, the main reduction rule is the same for both versions, thus with the present methods it seems difficult to obtain a better problem kernel for the "non-exact" version. In the following, we assume that the graph is stored using adjacency lists.

A central ingredient for the problem kernel is the notion of a clean region.

Definition 5.1. *We call a vertex of G* clean *if it has degree d, and* dirty *otherwise. We define a* clean region *in G as a maximal subset of clean vertices that induces a connected subgraph in G.*

Let $\{C_i \mid i \in I\}$ be the set of all clean regions. The neighborhood of each clean region C_i is called its *boundary* B_i, that is, $B_i := N(C_i)$. A clean region C_i is called *isolated* if $B_i = \emptyset$. Observe that the neighborhood of a non-isolated clean region consists entirely of dirty vertices. See Figure 5.2 for examples of clean regions and their boundaries. The detection of all clean regions in G can be done in $O(dn)$ time using a modified breadth-first search approach that does not search "beyond" dirty vertices.

5.2.1 Data Reduction Rules

In this section, we give a series of polynomial-time executable data reduction rules; after that, we show that these data reduction rules lead to a problem

kernel of the claimed size. The first reduction rule deletes vertices of too low and too high degree.

Reduction Rule 5.1. *If there exists a vertex $v \in V$ of degree less than d or of degree more than $d + k$, then delete v and decrease k by one.*

Lemma 5.1. *Reduction Rule 5.1 is correct and can be exhaustively applied in $O(n(k + d))$ time.*

Proof. A vertex v in G with $\deg(v) < d$ obviously must be contained in the solution S. Likewise, a vertex v with degree $\deg(v) > k + d$ must be in S, as we would have to put more than k of its neighbors into S to achieve degree d for v.

We briefly comment on the data structure supporting the implementation of Reduction Rule 5.1. First of all, we may assume from the beginning that $|E(G)| \leq n(k + d)$, because otherwise G contains a subgraph of minimum degree $> k + d$ and in this case we know in advance that the input graph is a no-instance. We construct an auxiliary data structure as follows. We create an array A of length n with entries from 0 to $n - 1$, where each entry i points to a linked list L_i. The entries of L_i correspond to the vertices of G that have degree i in G and contain pointers to these vertices in the adjacency list structure. Also, each vertex v in the adjacency list structure points back to the entry of $L_{\deg(v)}$ that points to it. This structure can be built on the top of the adjacency list structure in $O(|E(G)|) = O(n(k + d))$ time. It is now straightforward to verify that, using this enhanced data structure, Reduction Rule 5.1 can be implemented to run in $O(|E(G)|) = O(n(k + d))$ time. □

After Reduction Rule 5.1 has been applied exhaustively, the vertices in G have maximum degree $d + k$. Thus, assuming that a solution S of size k exists, we know that its neighborhood $D := N(S)$ has size at most $k(d + k)$. However, we still cannot bound the number of the remaining vertices $F := V \setminus (S \cup N(S))$, because there might be arbitrary big clean regions in the graph. See Figure 5.3 for an illustration of the corresponding structure of the graph. The remaining data reduction rules deal with the clean regions in F.

Observe that taking a vertex of a clean region into the solution S causes its clean neighbors to have a degree less than d in $G - S$, forcing them into the solution as well. By applying the same argument inductively to the clean neighbors, we can see that either no vertex of a clean region is a part of the solution S, or the entire clean region is contained in S. This observation is needed for the next data reduction rule.

Reduction Rule 5.2. *If G contains a clean region C_i whose boundary $B_i = N(C_i)$ contains a vertex with more than d neighbors in C_i, then delete all vertices in C_i from G and decrease k by $|C_i|$.*

Lemma 5.2. *Reduction Rule 5.2 is correct and can be exhaustively applied in $O(dn)$ time.*

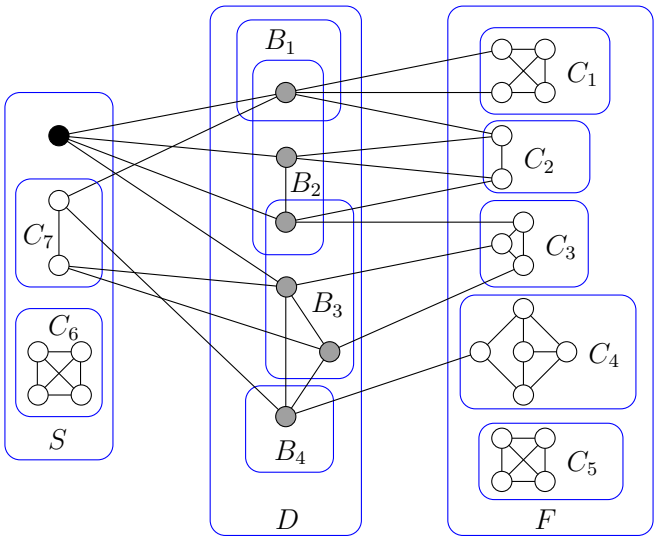


Figure 5.3: Solution S for $d = 3$ and $k = 7$, its neighborhood $D := N(S)$ (consisting entirely of dirty vertices), and the remaining vertices $F := V \setminus (S \cup D)$ (consisting entirely of clean vertices). Dirty vertices are gray or black; boundary vertices are gray and all other dirty vertices are black. The boundary of the clean region C_7 in S is not labeled. The isolated clean region C_6 is part of the solution S because we are considering the "exact" version of Regular-Degree-d Vertex Deletion.

Proof. For the correctness, assume that a solution S of size exactly k exists, that is, $G - S$ is d-regular. All vertices in $N(S)$ are dirty, since they have degree exactly d in $G - S$, and hence degree greater than d in G. Therefore, a clean region is a subset of either S or $V(G) \setminus (S \cup N(S))$.

We show that if there is a vertex $v \in B_i$ with $|N(v) \cap C_i| > d$, then C_i is a subset of S. To this end, suppose that there exists a vertex $v \in B_i$ with $|N(v) \cap C_i| > d$ and $C_i \cap S = \emptyset$. This means that v must be in S, but then the neighbors of v in C_i do not have degree d in $G - S$, a contradiction. Therefore $C_i \subseteq S$ and hence Reduction Rule 5.2 is correct.

We now comment on the running time of Reduction Rule 5.2. First, we find all clean regions by a modified breadth-first search in $O(dn)$ time. The clean regions are stored as linked lists. For each clean region, we go through all vertices and their adjacency lists and, by maintaining a counter for each vertex, we count for every dirty vertex in the boundary how many neighbors there are in this clean region. This takes $O(dn)$ time for all clean regions. If in this process we find a dirty vertex whose counter exceeds r, then we delete the corresponding clean region. This takes $O(dk)$ time in total, since we can delete at most k vertices, each having degree d. Therefore, the running time of exhaustively applying Reduction Rule 5.2 is dominated by $O(dn)$. □

In the following, we assume that G is reduced with respect to Reduction

Rules 5.1 and 5.2. Our observation that either no vertex of a clean region or the entire clean region is contained in the solution implies that isolated clean regions that contain more than k vertices cannot be part of the solution. This leads to the next two reduction rules, which deal with isolated clean regions in the graph. Recall that we are solving the exact version of Regular-Degree-d Vertex Deletion, that is, we are demanding for a solution of size exactly k. For this reason, we cannot just delete every isolated clean region in the graph.

Reduction Rule 5.3. *If G contains an isolated clean region C_i of at least $k+1$ vertices, then delete all vertices in C_i from G.*

This data reduction rule is obviously correct and can be performed in $O(dn)$ time.

Reduction Rule 5.4. *For $i = d+1, \ldots, k$ do: if G contains s isolated clean regions of i vertices, then modify G by deleting all but $\lfloor k/i \rfloor$ of them.*

Lemma 5.3. *Reduction Rule 5.4 is correct an can be exhaustively applied in $O(dn)$ time.*

Proof. An optimal solution S can contain at most $\lfloor k/i \rfloor$ clean regions of size i; additional clean regions of size i are superfluous and can therefore be safely deleted.

We can find all isolated clean regions by a modified breadth-first search in $O(dn)$ time. Build an array of length $k-d$ with entries from $d+1$ to k, where entry i points to a (linked) list of all clean regions containing i vertices. With the help of this array, it is straightforward to find (in $O(k)$ time) and delete (in $O(dn)$ time) all clean regions that meet the condition of the reduction rule. Thus, $O(dn)$ dominates the running time required for Reduction Rule 5.4. □

The idea for the next reduction rule is to replace big non-isolated clean regions that contain more than k vertices by smaller ones, which have a size bounded by a function of k, but contain still more than k vertices in order to get an equivalent problem instance. In this process, the degree of the vertices in the boundary of the corresponding clean regions must not change. For $d = 1$ this step does not apply, as then each non-isolated clean region contains exactly one vertex. For $d = 2$, the task is essentially just to replace long paths by shorter ones, which can be easily dealt with (we do that later in the description of the next reduction rule). The replacement gets more involved for $d \geq 3$, as we generally must be able to give an appropriate regular gadget with the constraints mentioned above. We describe the technical details of the replacement in what follows, then we give the reduction rule and, after that, we show its correctness.

We need to consider the set of edges between a clean region and its boundary. Let E_i be the set of edges connecting vertices in B_i with vertices in C_i. We search for all clean regions C_i of size greater than $(x+1)(d+1)$, where $x := \max\{|B_i|, \lceil (k+1)/(d+1) \rceil\}$, and replace each one by a new clean region of size $(x+1)(d+1)$ without changing the degrees of any vertex in the corresponding

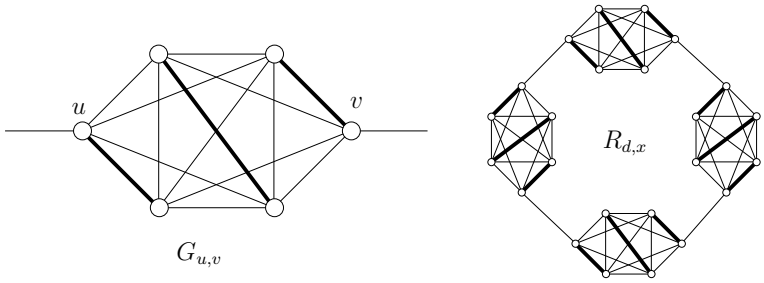


Figure 5.4: The graph $G_{u,v}$ for $d = 5$ and a matching (bold edges) in it and the corresponding graph $R_{d,x}$ for $x = 3$ containing four copies of $G_{u,v}$.

boundary B_i (notice that $(x+1)(d+1) \geq x(d+1) \geq k+1$, which is important as this prevents such a new clean region from being part of the solution). We first describe the gadget needed for the replacement, then we state the reduction rule.

We replace a clean region C_i by a d-regular structure $R_{d,x}$ of $(x+1)(d+1)$ vertices, and then reconnect the vertices in B_i with vertices in $R_{d,x}$ such that $R_{d,x}$ remains clean (for this, we apply some modifications in $R_{d,x}$) and such that the degree of the vertices in B_i is as before the replacement. The d-regular structure $R_{d,x}$ is constructed as follows. Take a cycle of $2(x+1)$ edges, remove every second edge $\{u, v\}$ and replace it by a graph $G_{u,v}$ consisting of a $(d-1)$-clique whose vertices are all connected with v and u (notice that $G_{u,v}$ is K_{d+1} with an edge removed). See Figure 5.4 for an example of such a graph $G_{u,v}$ and the entire d-regular gadget $R_{d,x}$. The resulting graph is d-regular and contains $(x+1)(d+1)$ vertices.

In the reduction step we reconnect vertices in B_i with vertices in $R_{d,x}$ by removing some edges in $R_{d,x}$ and then connecting their endpoints with vertices in B_i. In this process, we must assure that the new clean region does not decay into several clean regions. Therefore, we define a set of edges M in $R_{d,x}$ such that the graph which results by removing M from $R_{d,x}$ consists of exactly one connected component: Any single $G_{u,v}$ contains a matching $M_{u,v}$ of size $\lceil d/2 \rceil$, which can be constructed by using a Hamiltonian path from u to v[1], removing every second edge on it. See Figure 5.4 for an example. Let M be the union of all $M_{u,v}$. Observe that M is a matching of size $(x+1)\lceil d/2 \rceil$ for $R_{d,x}$, since there are $x+1$ copies of $G_{u,v}$ in $R_{d,x}$. Furthermore, observe that $R_{d,x}$ remains connected if we remove from it all edges of M, since in each $G_{u,v}$ there is always a u-v-path that contains no edge from $M_{u,v}$. (this property is independent of how the set $M_{u,v}$ was chosen for each $G_{u,v}$). The next reduction rule applies these ideas (see also Figure 5.5).

Reduction Rule 5.5. *For $d = 2$, if there exists a clean region C_i, which forms a path, with endpoints a and b in G such that $|C_i| > k+1$, then apply the following*

[1]Such a path can be found simply by starting in u and then visiting each vertex such that the last vertex is v—recall that $G_{u,v}$ is a K_{d+1} with one edge removed.

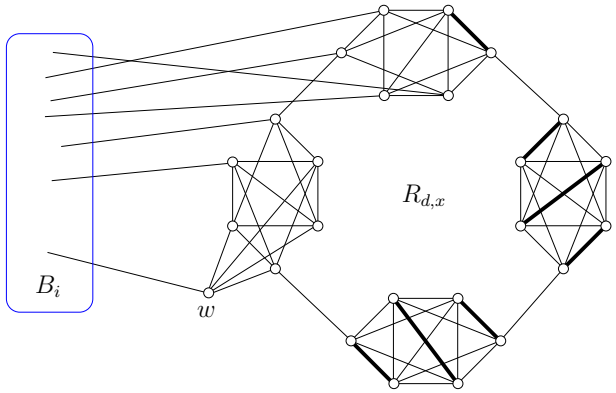


Figure 5.5: Example of how Reduction Rule 5.5 uses the gadget $R_{d,x}$ to replace a clean region which was connected to B_i by seven edges. The example corresponds to the most involved case 4.2 of the replacement procedure.

replacement procedure. Remove all vertices in $C_i \setminus \{a, b\}$ *from* G *and reconnect the endpoints* a *and* b *of* C_i *by a path with* $k - 1$ *new vertices.*

For $d \geq 3$*, if there exists a clean region* C_i *in* G *such that* $|C_i| > (x+1)(d+1)$*, where* $x := \max\{|B_i|, \lceil (k+1)/(d+1) \rceil\}$*, then apply the following replacement procedure.*

1. *Subdivide in* G *all edges in* E_i*. Let* L *be the set of subdivision vertices.*

2. *Remove all vertices in* C_i *from* G*.*

3. *Add* $R_{d,x}$ *to* G*, that is, set* $G := (V(G) \cup V(R_{d,x}), E(G) \cup E(R_{d,x}))$*.*

4.1 *If* $|L|$ *is even, then choose, arbitrarily, a subset* $M' \subseteq M$ *where* $|M'| = |L|/2$*, and remove the edges of* M' *from* $R_{d,x}$*. Identify, arbitrarily, their endpoints with the vertices in* L*.*

4.2 *If* $|L|$ *is odd, then choose, arbitrarily, a subset* $M' \subseteq M$ *where* $|M'| = (|L| + d - 2)/2$*, and remove the edges of* M' *from* $R_{d,x}$ *(as we will see,* d *is also odd in this case). Identify, arbitrarily,* $|L| - 1$ *of their endpoints with all vertices in* L *except one (say* w*), and then make* w *adjacent with the remaining* $2|M'| - (|L| - 1) = |L| + d - 2 - (|L| - 1) = d - 1$ *endpoints of the edges in* M'*.*

See Figure 5.5 for an example of how the replacement works.

Lemma 5.4. *Reduction Rule 5.5 is correct and can be exhaustively applied in* $O(dn)$ *time.*

Proof. First, we show that the reduction rule replaces a clean region C_i of size more than $(x + 1)(r + 1)$ by one of size at least $k + 1$. Then, we show that

this replacement always yields an equivalent problem instance. After that, we describe how the claimed running time can be achieved.

For $d \geq 2$ the replacement is clearly correct, as clean regions with more than $k+1$ vertices are replaced by clean regions with exactly $k+1$ vertices. To see that this procedure works correctly for $d \geq 3$, consider the following remarks corresponding to the steps of the reduction rule.
(1.) Since G is reduced with respect to Reduction Rule 5.2, we have $|E_i| \leq d|B_i|$ and hence $|L| \leq d|B_i|$.
(4.) The choice of a set M' of claimed size is always possible since the set M is large enough. More formally, we verify that

$$\begin{aligned} |M| &= (x+1) \cdot \lceil d/2 \rceil \\ &\geq (x+1) \cdot d/2 \\ &\geq (|B_i|+1) \cdot d/2 \\ &\geq (|L|+d)/2 > |M'|. \end{aligned}$$

(4.1.) After removing the edges in M', its endpoints have degree $d-1$. However, after identifying each such endpoint with one vertex in L, all vertices in $R_{d,x}$ have degree d again, since M is a matching. Due to the properties of M mentioned above, $R_{d,x}$ is still one clean region.
(4.2.) First of all, note that $(|L|+d-2)/2$ is a positive integer, since $|L| \geq 1$ and since $|L| = |E_i|$ being odd implies d to be odd; this can be easily seen by analyzing the number of edges in $G[C_i]$ and between C_i and B_i as $2|E'| + |E_i| = d|C_i|$, where E' is the set of edges in $G[C_i]$. With the method in (4.1.) we can only identify an even number of vertices in L with vertices in $R_{d,x}$. Therefore, there remains one vertex in L which has to be made adjacent to the $(d-1)$ remaining endpoints of edges in M'. It is easy to see that afterwards all vertices in $R_{d,x}$ have degree d and that $R_{d,x}$ is still one clean region.

Thus, for $d \geq 3$, Reduction Rule 5.5 replaces a clean region C_i of size more than $(x+1)(d+1)$ by a clean region C_i' of size $(x+1)(d+1) \geq k+1$. Moreover, the new clean region C_i' is connected, has the same boundary B_i as C_i, and all vertices in B_i have the same number of neighbors in C_i' as they had in C_i.

Next, we prove that Reduction Rule 5.5 returns an equivalent instance for Regular-Degree-d Vertex Deletion. For this, let S be a size-k vertex set such that $G - S$ is r-regular, and let G' be the graph after applying Reduction Rule 5.5 to region C_i in G. A solution S for G cannot contain a vertex of any C_i that changed, as C_i contains more than k vertices. We retained the vertex degree of all vertices in B_i, C_i' is also a clean region in G', and we did not alter the subgraph $G - C_i = G' - C_i'$, thus $G' - S$ is also r-regular. Therefore, S is also a solution for G'. The same argument holds for the other direction: A solution S' for G' cannot contain any vertex of any C_i', as C_i' contains more than k vertices, and since $G - C_i = G' - C_i'$ we know that S' is also a solution for G.

The implementation of this reduction rule again follows the ideas of the previous ones. Again, clean regions can be detected by a modified breadth-first

search in $O(dn)$ time. Note that the number of edges to be subdivided in the whole graph is at most dn. Subdividing the edges takes $O(dn)$ time, and removing clean regions takes $O(dn)$ time in total. The new clean regions can have at most n vertices and $O(dn)$ edges in total, thus the total running time for this step is again $O(dn)$. □

5.2.2 Kernel Size

In the following, we assume that the reduction rules in Section 5.2.1 have been applied in the given order. First, we show that the number of isolated clean regions is bounded.

Lemma 5.5. *After Reduction Rules 5.3 and 5.4 have been applied exhaustively, the resulting graph contains at most k^2 vertices in isolated clean regions.*

Proof. Reduction Rule 5.3 deletes all isolated clean regions of at least $k+1$ vertices. Reduction Rule 5.4 deletes all but $\lfloor k/i \rfloor$ isolated clean regions of equal size i, for each $d+1 \leq i \leq k$. Considering all possible sizes of clean regions in the remaining graph (at most k), we can conclude that there are at most

$$\sum_{i=1}^{k} \lfloor k/i \rfloor \cdot i \leq \sum_{i=1}^{k} k = k^2$$

vertices in isolated clean regions. □

Summarizing, the reduced graph satisfies a bigger subset of the following properties:

(1) All vertices in G have degree at least d and at most $k+d$ (Reduction Rule 5.1),

(2) each vertex of a boundary B_i has at most d clean neighbors in C_i (Reduction Rule 5.2),

(3) the isolated clean regions of G contain in total at most k^2 vertices (Reduction Rules 5.3 and 5.4),

(4) for every clean region C_i with boundary B_i,

$$|C_i| \leq (d+1)(1 + \max\{\lceil \frac{k+1}{d+1} \rceil, |B_i|\})$$

(Reduction Rule 5.5).

It remains to show the following.

Lemma 5.6. *Let (G,k) be an instance of* Regular-Degree-d Vertex Deletion. *If G satisfies properties (1)–(4) and contains more than $dk(k+d)(k+3d+4)+k+k(k+d)+k^2$ vertices, then (G,k) is a no-instance.*

Proof. Assume that a solution S of size exactly k exists, i.e., $G - S$ is d-regular. We define $D = N(S)$ and $F = V(G) \setminus (S \cup D)$ and observe that S, D, F is a 3-partition of $V(G)$. Due to property (1) every vertex in G has degree at most $d + k$. Therefore, the number of vertices in the neighborhood of S cannot exceed $k(d + k)$ and thus $|D| \leq k(d + k)$. We also observe that all vertices in F are clean, otherwise $G - S$ would contain a vertex not having degree d, which contradicts S being a solution. It remains to bound the size of F. Recall that a clean region C_i is either completely contained in S or no vertex in C_i is a member of S, thus $C_i \subseteq F$. Therefore, as all vertices in F are clean, F is a union of clean regions. Suppose that F consists of a set $\mathcal{C} = \{C_i \mid 0 \leq i \leq q\}$ of q non-isolated clean regions. As there is no edge in G between S and F (i.e., D separates S and F) and all vertices in F are clean, we obtain that all boundary vertices of the clean regions in $\mathcal{C}$ must be in D, i.e., $\bigcup_{i=1,\ldots,q} B_i \subseteq D$. Also, since $G - S$ is d-regular, each vertex of D belongs to at most d sets in $\mathcal{B} = \{B_1, \ldots, B_q\}$, and, therefore, $\sum_{i=1,\ldots,q} |B_i| \leq d \cdot |D| \leq dk(k + d)$. By property (4),

$$\begin{aligned} |C_i| &\leq (\max\{\lceil \frac{k+1}{d+1} \rceil, |B_i|\} + 1)(d + 1) \\ &\leq \max\{k + d + 2, |B_i| \cdot (d + 1)\} + d + 1. \end{aligned}$$

Recall that F contains at most $\sum_{i=1,\ldots,q} |C_i|$ vertices from non-isolated clean regions. By property (3), no more than k^2 vertices are contained in isolated regions. Therefore,

$$\begin{aligned} |F| &\leq k^2 + \sum_{i=1}^{q} (\max\{|B_i| \cdot (d + 1), k + d + 2\} + d + 1) \\ &\leq k^2 + \sum_{i=1}^{q} (|B_i| \cdot (d + 1) + k + 2d + 3) \\ &\leq k^2 + \sum_{i=1}^{q} (|B_i| \cdot (d + 1)) + \sum_{i=1}^{q} (k + 2d + 3) \\ &= k^2 + dk(k + d)(d + 1) + dk(k + d)(k + 2d + 3) \\ &= k^2 + dk(k + d)(k + 3d + 4). \end{aligned}$$

Thus, if there exists a solution S, since $|S| = k$ and $|D| \leq k(k + d)$ we can conclude that G can have at most $dk(k + d)(k + 3d + 4) + k + k(k + d) + k^2$ vertices. The contra position of this argument shows the lemma. □

To complete the proof of Theorem 5.1, recall that for all data reduction rules described in Section 5.2.1, the running time was no worse than the running time of constructing an enhanced data structure (see proof of Lemma 5.2), which is $O(|E(G)|) = O(n(k + d))$. The kernel size follows directly from Lemma 5.6. For $r = 1$, every non-isolated clean region contains a single vertex and Reduction

Rule 5.5 does not apply at all. This allows for a better counting of the vertices in F (proof of Lemma 5.6), which, apart from those belonging to isolated clean regions, are at most as many as the vertices in D. As any isolated clean region contains exactly two vertices when $r = 1$, Reduction Rule 5.4 is applied only for $i = 2$, leaving at most $k + 1$ vertices in isolated clean regions. Therefore, in the case $r = 1$, the kernel has size at most $|S|+2|D| \leq k+2k(k+1)+k+1 = O(k^2)$.

Remark. Notice that Theorem 5.1 holds also for the "non-exact" version (demanding a solution of size at most k) of Regular-Degree-d Vertex Deletion. The only modification is that we have to replace Reduction Rule 5.4 by the deletion of all isolated clean regions. However, the main "power" for reducing the graph comes from Reduction Rule 5.5; therefore, it seem difficult to obtain a better kernel size for the "non-exact" version.

5.3 Search Tree Algorithm

In this section, we describe a simple exact algorithm for Regular-Degree-d Vertex Deletion running in $O(n(k + d) + (d + 2)^k)$ time (working for the "exact" and "non-exact" version). It is based on a bounded search tree technique. Let (G, k) be an instance of Regular-Degree-d Vertex Deletion. We first describe the "non-exact" version, we then state the differences for the "exact" version below. While the graph G is not d-regular and $k > 0$, we choose an arbitrary vertex $v \in V(G)$ with $\deg(v) > d$ and branch into the following cases, where in each case we also always exhaustively apply Reduction Rule 5.1 from Section 5.2.1 (here, we would actually only need a rule that deletes vertices of degree less than d).

1. If v is a part of the solution, then delete v from G and decrease k by one.

2. If v is not a part of the solution (thus it remains in G), then choose an arbitrary subset of $d + 1$ neighbors of v. At least one of these neighbors must be contained in the solution in order to achieve degree d for v; thus we branch into the $d + 1$ subcases, and, in each of the subcases, we choose a neighbor, delete it from G, and decrease k by one.

Observe that, unlike for the similar search tree algorithm for Bounded-Degree-d Vertex Deletion in Section 4.4.1 of Chapter 4, the application of a data reduction rule that deletes low-degree vertices is necessary; otherwise, vertices of degree less than d would never be deleted from the graph.

For the "exact" version, the algorithm branches until G is d-regular and $k = 0$. That is, unlike the "non-exact" version, if the algorithm encounters a d-regular graph, but $k > 0$, then it does not return but keeps on branching.

In each subcase of the branching we put at least one vertex from G into the solution, and we branch into $d + 2$ subcases. This results in a search tree

of size $O((d+2)^k)$. The test of d-regularity can be done in $O(dn)$ time. This means that a solution for Regular-Degree-d Vertex Deletion can be found in $O(dn(d+2)^k)$ time, if it exists. Combining this with Theorem 5.1 and using an improved interleaving method [NR00], we arrive at the following result.

Theorem 5.2. *For any $d \geq 0$, there exists an algorithm for* Regular-Degree-d Vertex Deletion *with parameter k that runs in $O(n(k+d)+(d+2)^k)$ steps.*

5.4 Hardness Results

In this section, we prove that Regular-Degree-d Vertex Deletion is NP-complete by giving a polynomial-time many-one reduction from Vertex Cover. As the "non-exact" version is weaker than (can be reduced to) its "exact" counterpart, we give the hardness proof for the "non-exact" version. A similar independent result is given by Cardoso et al. [CKL07], proving the NP-hardness of finding an induced r-regular (bipartite) graph.

We give a proof that Regular-Degree-d Vertex Deletion and therefore also its dual parameterization d-Regular Induced Subgraph are NP-complete. After that, we shortly describe how the proof can also be used to show the W[1]-hardness of d-Regular Induced Subgraph.

Theorem 5.3. Regular-Degree-d Vertex Deletion *is NP-complete for every $d \geq 0$. It remains NP-complete when restricted to planar graphs (for $d \leq 5$) or to triangle-free planar graphs (for $d \leq 3$).*

Proof. We first prove the theorem in its general statement and then we explain how to modify the proof for its planar versions. For $d = 0$, Regular-Degree-d Vertex Deletion is identical to Vertex Cover, which is known to be NP-complete [GJ79]. For all remaining $d > 0$ we give a reduction from Vertex Cover.

Let (G, k) be an instance of Vertex Cover. We construct an instance (G', k') of Regular-Degree-d Vertex Deletion with $d > 0$ as follows: First, we set $G' := G$ and $k' := k(d+1)$. For each vertex $v \in V(G)$ we add a copy of a $(d+1)$-vertex clique K_{d+1} to G'. Let R_v be the copy of K_{d+1} corresponding to vertex v. For all vertices $v \in V(G)$ we identify v with an arbitrary vertex in R_v, that is, we set $v = w$ for some arbitrary $w \in V(R_v)$. Figure 5.6 gives an example of a graph G and the corresponding graph G'.

We have to show that (G, k) with $d = 0$ is a yes-instance if and only if (G', k') with $d > 0$ is a yes-instance.
($\Rightarrow$) Suppose that there is a size-k solution S for (G, k), that is, $G - S$ consists of isolated vertices. We define a new solution set $S' := \bigcup_{v \in S} V(R_v)$ of size $k \cdot (d+1)$ for G'. Clearly, $G' - S'$ is a graph in which every connected component is an R_v, i.e., $G' - S'$ is a d-regular graph, thus S' is a solution for (G', k').

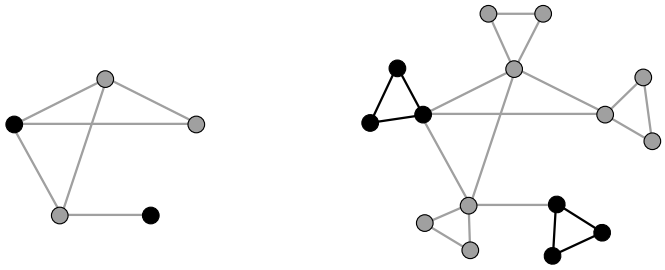


Figure 5.6: Example of a graph G (left) and the corresponding graph G' (right) with $d = 2$. Vertices in the solution are gray, the remaining vertices which are not deleted are black.

($\Leftarrow$) Suppose that there is a size-k' solution S' for (G', k'). We say that S' is *clustered* if

$$\forall v \in V(G) : R_v \cap S' \neq \emptyset \Rightarrow R_v \subseteq S'.$$

Notice that if S' is clustered, then $S := \{v \in V(G) \mid R_v \cap S' \neq \emptyset\}$ is a solution for the instance (G, k) of Vertex Cover. In the case that the solution S' is not clustered, we can turn it into a clustered one according to the following claim, which completes the proof of correctness of the reduction.

Claim: *Given a solution S' for (G', k') where $|S'| \leq k'$, we can always construct a clustered solution S'' for the same problem instance.*

Proof of Claim: We first show that $G' - S'$ is a d-regular graph in which each connected component is either a R_v or a d-regular subgraph containing vertices exclusively from G. Recall that $G' - S'$ is d-regular. If $v \in S'$ for some $v \in G$, then we know that $R_v \subseteq S'$, as otherwise some vertices in R_v would have degree smaller than d in $G' - S'$. The same argument shows that if $v \notin S'$, then either $R_v \cap S' = \emptyset$ or $(R_v - v) \subseteq S'$. The first case implies that every neighbor w of v in G is in the solution S' (and therefore R_w likewise), since otherwise v would have a degree greater than d in $G' - S'$. The second case implies that d neighbors of v in G must not be in the solution S', since otherwise v would have a degree smaller than d in $G - S'$.

With these observations we can prove the claim as follows. Assume that $G' - S'$ contains some connected components which are subgraphs only consisting of vertices from G. Let A be the set of vertices of all such connected components of $G' - S'$. As $G'[A]$ is a d-regular graph, it will contain an independent set I of size at least $\lceil |A|/(d+1) \rceil$ (I is constructed by greedily picking vertices and removing their neighbors). We set $S'' = (S' \cup (A \setminus I)) \setminus \{V(R_v) \mid v \in I\}$ and we observe that $G' - S''$ is also a d-regular graph where each connected component is an R_v. To show that S'' is a solution for (G', k'), it remains to prove that $|S''| \leq k$. Observe that the above modification added at most $|A| - \lceil |A|/(d+1) \rceil$ vertices to the solution and deleted at least $d \cdot \lceil |A|/(d+1) \rceil$ vertices from it. Using the tautological relation $|A|/(d+1) \leq \lceil |A|/(1+d) \rceil$, which can be rewritten

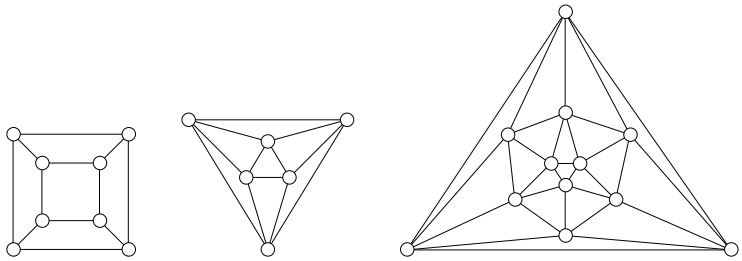


Figure 5.7: (a) Cubical, (b) octahedral, and (c) icosahedral graph, which describe the connectivity of the vertices of a cube, an octahedron, and an icosahedron, respectively.

as $|A| - \lceil |A|/(d+1) \rceil - d \cdot \lceil |A|/(d+1) \rceil \leq 0$, we get

$$|S''| - |S'| \leq |A| - \lceil \frac{|A|}{d+1} \rceil - d \cdot \lceil \frac{|A|}{d+1} \rceil \leq 0.$$

Hence $|S''| \leq |S'| \leq k$, and this completes the proof of the claim.

Vertex Cover remains NP-complete when restricted to triangle-free planar graphs [GJS76]. We obtain the NP-completeness of Bounded-Degree-d Vertex Deletion on planar graphs by reducing from Vertex Cover on triangle-free planar graphs and modifying the gadget R_v slightly: for $d \leq 3$, we use as R_v the 3-regular cubical graph (Figure 5.7a) instead of K_4 in the construction. For $d = 4$ and $d = 5$, we use as R_v the 4-regular octahedral graph (Figure 5.7b) and the 5-regular icosahedral graph (Figure 5.7c) instead of K_5 and K_6 in the construction, respectively. Therefore, the above proof also implies that Regular-Degree-d Vertex Deletion is NP-complete even when restricted to planar graphs for $d \leq 5$. Moreover, for $d \leq 3$, Bounded-Degree-d Vertex Deletion is NP-complete even on triangle-free planar graphs. □

It is known that the parameterized version of Independent Set (the dual parameterization of Vertex Cover), where the parameter is the size of the independent set, is W[1]-hard [DF95]. The reduction in the above proof can be used to show the W[1]-hardness of d-Regular Induced Subgraph as follows. Let (G, k) be an instance of Independent Set. It can be regarded as an instance $(G, n-k)$ of Vertex Cover. This instance is reduced to an instance $(G', (n-k)(d+1))$ of Regular-Degree-d Vertex Deletion, which can be regarded as an instance $(G', n(d-1)-(n-k)(d+1)) = (G', k(d+1))$ of d-Regular Induced Subgraph. Clearly, this is a parameterized reduction, as all these steps can be performed in polynomial time, and the parameter $k' = k(d+1)$ is only depending on k and d, where d is a constant. We arrive at the following theorem.

Theorem 5.4. *d-Regular Induced Subgraph is W[1]-hard with respect to parameter k.*

Table 5.2: Parameterized complexity of Regular-Degree Vertex Deletion (RDVD) and Bounded-Degree Vertex Deletion (BDVD) depending on d.

	RDVD	BDVD
d constant	FPT (Section 5.3)	FPT (Section 4.4)
d part of input	W[1]-hard [MS08a]	W[2]-complete (Section 4.6)
d unspecified	W[1]-hard [MS08a]	trivial

5.5 Outlook

In this chapter, we showed how to obtain a cubic-vertex problem kernel for Regular-Degree-d Vertex Deletion for constant d. In the construction of the kernel we used the fact that big "clean regions" can be safely replaced by smaller (but not too small) ones. Because r-regularity is not a hereditary property, we had to take care that such a replacement locally maintains r-regularity. Additionally, we gave a simple search tree algorithm for Regular-Degree-d Vertex Deletion and showed that its dual parameterization d-Regular Induced Subgraph is W[1]-hard with respect to parameter k.

Although Regular-Degree-d Vertex Deletion and Bounded-Degree-d Vertex Deletion (Chapter 4) seem to be quite similar in terms of computational and parameterized complexity for constant d, the picture changes for unbounded or unspecified d. If d is not given in advance, that is, that we ask that the resulting graph should be regular without specifying the degree, then Regular-Degree Vertex Deletion is W[1]-hard [MS08a] with respect to parameter k. Moreover, this variant of the problem is equivalent with respect to parameterized complexity to the variant where the desired degree d is a part of the input [MS08a]. However, Bounded-Degree Vertex Deletion is trivially solvable if the degree is not specified (just use the maximum degree of the input graph), but W[2]-complete with respect to parameter k if d is a part of the input (Theorem 4.6 in Section 4.6). Therefore, it would be interesting to see whether Regular-Degree Vertex Deletion is actually W[1]-complete or even harder (e.g., W[2]-complete). See also Table 5.2. In this respect, to study the differences between Regular-Degree-d Vertex Deletion and Bounded-Degree-d Vertex Deletion, we suggest to study the following problem that generalizes both problems.

> Degree-Range-(d_1, d_2) Vertex Deletion
> **Input:** An undirected graph $G = (V, E)$ and an integer $k \geq 0$.
> **Question:** Does there exist a vertex subset $S \subseteq V$ of size at most k such that $G - S$ has degree at least d_1 and at most d_2?

Obviously, for $d_1 = 0$ and $d_2 = d$ we obtain Bounded-Degree-d Vertex Deletion, and for $d_1 = d_2 = d$ we obtain Regular-Degree-d Vertex Deletion.

The results in this chapter also have some practical potential. Regular-Degree-d Vertex Deletion for $d = 1$ is the dual of Induced Matching (a relatively hard problem with many applications, see Chapter 8). This relation could be used to solve Induced Matching in practice via transformation to Regular-Degree-1 Vertex Deletion, which is then solved using the problem kernel and the search tree algorithm presented in this chapter. An analogous approach that exploits a similar dual relation between Bounded-Degree Vertex Deletion (Chapter 4) and the (notoriously hard) problem of finding maximum s-plexes (Chapter 9) is used to find maximum s-plexes efficiently in many real-world graphs.

Chapter 6

Vertex Deletion Problems and Iterative Compression

With the introduction of the iterative compression technique by Reed, Smith, and Vetta [RSV04], parameterized complexity analysis has gained a new tool for showing fixed-parameter tractability results for NP-hard minimization problems (cf. Section 2.3.3). Many of these problems are vertex deletion problems for hereditary graph properties. We investigate the computational complexity of a general "compression task" centrally occurring in all known applications of iterative compression. The core issue (particularly but not only motivated by iterative compression) is to determine the computational complexity of, given an already inclusion-minimal solution for an underlying (typically NP-hard) vertex deletion problem in graphs, finding a better *disjoint* solution. The complexity of this task so far has been lacking a systematic study. We consider a large class of vertex deletion problems for hereditary graph properties on undirected graphs and show that, except for few cases which are polynomial-time solvable, the others are NP-complete. This class includes problems such as Vertex Cover and Bounded-Degree-1 Vertex Deletion (cf. Section 4.5), where the corresponding compression task is solvable in polynomial time or Undirected Feedback Vertex Set (here the corresponding compression task is NP-complete).

We first give a short overview on the known applications of the iterative compression technique (Section 6.1). Then, we outline an iterative compression framework for vertex deletion problems for hereditary properties (Section 6.2). After that, we present a complexity dichotomy for the disjoint compression task, that is, we completely classify the disjoint compression tasks of the considered vertex deletion problems with respect to their computational complexity (Section 6.3).

6.1 Known Results

Using iterative compression, it has been shown that several NP-hard *feedback set problems*, that is, problems where the task is to destroy certain cycles with at most k vertex or edge deletions, are fixed-parameter tractable with respect to the parameter k:

- Vertex Bipartization: Make a graph bipartite by destroying all odd-length cycles with at most k vertex deletions [RSV04, Hüf09, LSS09]. The best known running time is $O(3^k \cdot mn)$ [Hüf09].
- Edge Bipartization: Make a graph bipartite by destroying all odd-length cycles with at most k edge deletions; the running time is $O(2^k \cdot m^2)$ [GGH+06].
- Undirected Feedback Vertex Set: Destroy all cycles with at most k vertex deletions [DFL+07, GGH+06, CFL+08]. The best known running time is $O(5^k \cdot kn^2)$ [CFL+08].
- Directed Feedback Vertex Set: Destroy all cycles in a directed graph with at most k vertex deletions; the running time is $O(k! \cdot 4^k \cdot k^3 n^4)$ [CLL+08].
- Directed Feedback Vertex Set in tournaments: Destroy all cycles in a tournament[1] with at most k vertex deletions; the running time is $O(2^k \cdot n^2(\log \log n + k))$ [DGH+09].
- Chordal Deletion: Make a graph chordal by destroying all chordless cycles[2] with at most k vertex deletions [Mar09]. No explicit running time is stated.
- Signed Graph Balancing: Make a signed graph balanced by destroying all unbalanced cycles with at most k edge deletions [HBN09]. This problem can be reduced to Edge Bipartization, yielding a running time of $O(2^k \cdot m^2)$ [GGH+06].

In the following, we turn our attention to the vertex deletion problems in the above list. The desired graph property of each of these problems is hereditary. This makes the problems particularly amenable to iterative compression (this will become clear later in Section 6.2). For each of these graph properties except for Directed Feedback Vertex Set in tournaments, the set of forbidden induced subgraphs is infinite (basically cycles of all possible lengths). Therefore, it is not immediately clear that the corresponding vertex deletion problem is fixed-parameter tractable (cf. [Cai96]). For Directed Feedback Vertex Set in

[1] A *tournament* is a directed graph whose underlying undirected graph is complete, that is, there is exactly one directed edge between every pair of vertices.

[2] A *chordless cycle* is an induced cycle of length at least four.

tournaments, the benefit of using iterative compression is its simplicity and the improved running time compared to the best-known algorithm, which is based on a reduction to 3-Hitting Set; it can be solved in $O(2.076^k \cdot \text{poly}(n)))$ with a rather complicated bounded search tree approach [Wah07].

The strength of iterative compression for these types of problems lies in the compression task (cf. Section 2.3.3). Take Undirected Feedback Vertex Set as an example. A solution S for a graph G which is close to be optimal reveals a lot of information about the graph structure. In particular, $G - S$ is free of cycles, that is, $G - S$ is a forest. Finding a smaller solution with the help of such a forest turns out to be a much simplified task, which the most recent algorithm for Undirected Feedback Vertex Set solves with a bounded search tree approach [CFL$^+$08].

Iterative compression has also been applied to other types of problems.

- Vertex Cover: Destroy all edges of a graph by at most k vertex deletions [Guo06, Pei07] (cf. Section 6.3.1). These algorithms are not competitive with other (mostly search tree based) approaches, but are simple examples of how the iterative compression technique can be applied. The best known running time based on iterative compression is $O(1.443^k \cdot mn\sqrt{n})$ [Pei07].

- Bounded-Degree-1 Vertex Deletion: Delete at most k vertices such that the resulting graph has degree at most one (Section 4.5).

- Cluster Vertex Deletion: Delete at most k vertices such that the resulting graph is a *cluster graph*, that is, each connected component forms a clique (Section 6.3.1).

- d-Hitting Set: For a given size-m family of subsets of an n-element ground set, where the size of each subset is at most d, choose at most k elements such that each subset contains at least one chosen element. The running times for $d = 3$, $d = 4$, and $d = 5$ are $O(2.274^k \cdot kn^2)$, $O(3.076^k + m)$, and $O(4.076^k + m)$, respectively [Hüf07]. As Hüffner [Hüf07] points out, the algorithm for $d = 3$ is not competitive with the best-known algorithm by Wahlström [Wah07], but can be very easily implemented when using one of the already available Vertex Cover implementations as a subroutine [ACF$^+$04, FKH04]. For $d = 4$ and $d = 5$, the iterative compression approach is faster than the previously fastest algorithm [Fer05] (but not for $d \geq 6$ due to an increasing polynomial overhead).

- Almost 2-Sat: Delete a minimum number of clauses to make a 2-CNF formula satisfiable; the running time is $O(15^k \cdot km^3)$, where m is the number of clauses in the 2-CNF formula [RO09]. This result also implies that Vertex Cover parameterized above the size of a maximum matching M,

that is, the task to find a vertex cover of size at most $|M| + k$ ("above guarantee parameterization") [MRS+07a], is fixed-parameter tractable with respect to the parameter k, because it can be transformed to Almost 2-Sat in $f(k) \cdot \text{poly}(n)$ time.

Recently, first attempts in linking iterative compression with exact algorithms have been made. Fomin et al. [FGK+08] have iterative compression based exact algorithms for Independent Set, a counting version of Hitting Set, and Maximum Induced Cluster Subgraph, the dual problem of Cluster Vertex Deletion (cf. Section 6.3.1).

Experimental Results. The Vertex Bipartization algorithm has been heuristically improved and implemented [Hüf07, Hüf09]. The experiments on data from computational biology show that iterative compression can outperform other methods by orders of magnitude. For example, an instance originating from computational biology with 102 vertices and 307 edges can be solved in 6248 seconds with an ILP approach, whereas an iterative compression approach runs in 1 second, which can be further improved by algorithmic tricks [Hüf09]. For the iterative compression algorithm for Signed Graph Balancing, experiments show that this approach has about the same running time as approximation algorithms, while producing exact solutions [HBN09].

6.2 Iterative Compression Framework and Compression Task

In this section, we give a general iterative compression framework for vertex deletion problems and describe the disjoint compression task for which we give a complexity dichotomy (that is, a complete classification of the polynomial-time solvable cases and the NP-hard cases) in Section 6.3. For this complexity dichotomy, we restrict our attention to vertex deletion problems for *hereditary* graph properties, that is, graph properties that are closed under vertex deletion. The general problem is defined as follows. Let Π be a hereditary graph property.

Π-Vertex Deletion
Input: An undirected graph $G = (V, E)$ and a nonnegative integer k.
Question: Is there a vertex subset $S \subseteq V$ of size at most k such that $G - S \in \Pi$?

For example, Undirected Feedback Vertex Set corresponds to the case that Π means "being cycle-free" whereas for Vertex Bipartization Π means "being free of odd-length cycles".

First, we show how to employ the compression routine. The principle is the same as in the example for Bounded-Degree-1 Vertex Deletion in Section 4.5, that is, we build up the graph vertex by vertex, while always applying

Algorithm: ITERATE (G, k)
Input: An undirected graph G and a nonnegative integer k.
Output: A Π-deletion set of size at most k, or "no-instance".

```
1 V' ← ∅
2 S ← ∅
3 while V' ≠ V do
4       select a vertex v ∈ V \ V'
5       V' ← V' ∪ {v}
6       S ← S ∪ {v}
7       S ← Compress(G[V'], S)
8       if |S| > k then return "no-instance"
9 return S
```

Figure 6.1: Pseudo-code of the algorithm to solve Π-Vertex Deletion via iterative compression. The pseudo-code of the algorithm Compress is given in Figure 6.3.

the compression routine to keep the solution small. In other words, the vertices are the augmentation elements.

Iteration. In the following, we call a solution for Π-Vertex Deletion a Π-*deletion set*. The pseudo-code of the iteration algorithm is given in Figure 6.1. We start with empty vertex subsets $V' = \emptyset$ and $S = \emptyset$ (lines 1 and 2); clearly, an empty set is a Π-deletion set for an empty graph. Iterating over all graph vertices, step by step we add one vertex $v \in V \setminus V'$ to both V' and S (lines 4–6). Then S is still a Π-deletion set for $G[V']$. In each step we try to find a smaller Π-deletion set for $G[V']$ by applying a compression routine (line 7). It takes the graph $G[V']$ and the Π-deletion set S for $G[V']$, and returns a smaller Π-deletion set for $G[V']$, or proves that S is optimal (by returning a Π-deletion set of the same size). Note that a smaller Π-deletion set for $G[V']$ is always a minimum one and has size $|S| - 1$, since $S \setminus \{v\}$ is a minimum Π-deletion set for $G[V' \setminus \{v\}]$, and a minimum Π-deletion set for $G[V']$ cannot be smaller than a minimum Π-deletion set for $G[V' \setminus \{v\}]$.[3] Therefore, it is a loop invariant that the compressed Π-deletion set S is a minimum-size Π-deletion set for $G[V']$. If $|S| > k$ (line 8), then we can conclude that G does not have a Π-deletion set of size at most k. Since eventually $V' = V$, we obtain a Π-deletion set of size at most k for G once the algorithm returns S (line 9).

Note that almost all known applications of iterative compression on graph problems with vertex subsets as solutions essentially build up the graph in this way for the following reason. For the iterative compression to work, it is important

[3] In other words, Π-Vertex Deletion behaves *monotonically* with respect to adding vertices. This follows from the fact that the graph property Π is hereditary.

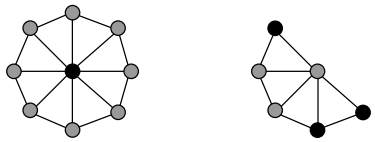


Figure 6.2: Example showing that Regular-Degree-d Vertex Deletion behaves non-monotonously with respect to vertex addition. For $k = 1$ and $d = 2$, the input graph G (left) can be made d-regular by deleting only the black vertex in the center. However, for an induced subgraph of G (right), a minimum-size solution (black vertices) contains $3 > k$ vertices.

that we can bound the size of an intermediate solution by k. This is usually accomplished by demanding that the problem behaves *monotonously* with respect to adding the augmentation element, that is, the intermediate solution must not become smaller by adding an augmentation element. In other words, a solution for a graph must not be smaller than a solution for any induced subgraph. Observe that a hereditary graph property Π immediately implies that the corresponding Π-Vertex Deletion problem behaves monotonously with respect to vertex addition. If a problem does not behave monotonously, then we cannot bound the size of an intermediate solution by k. For example, Regular-Degree-d Vertex Deletion (see Chapter 5), which is a vertex deletion problem for the non-hereditary graph property "d-regular", does not behave monotonously with respect to vertex addition (see Figure 6.2).

Compression. It remains to describe the compression routine. Given an undirected graph G and a solution S for Π-Vertex Deletion, the compression routine finds a smaller solution for G or proves that the solution S is of minimum size. As for our example Bounded-Degree-1 Vertex Deletion (Section 4.5), we reduce the compression routine to a *disjoint* compression routine by considering all partitions of S into one part to keep in the solution and one part to exchange. The compression routine works as follows. See Figure 6.3 for the corresponding pseudo-code. Consider a smaller Π-deletion set S' as a modification of the larger Π-deletion set S for the graph $G = (V, E)$. This modification retains some vertices $Y \subsetneq S$ as part of the solution set (that is, the vertices to be deleted), while the other vertices $X := S \setminus Y$ are replaced by new vertices from $V \setminus S$. The idea is to try by brute force all $2^{|S|} - 1$ nontrivial partitions of S into these two sets Y and X. For each such partition, the vertices from Y are immediately deleted from S (line 2) and G (line 3), since we already decided to take them into the Π-deletion set. If $G[X] \notin \Pi$, then we know that there can be no solution S' with $S' \cap S = Y$; hence, we only proceed if $G[X] \in \Pi$ (line 4). In the remaining instance $G' := G - Y$, it remains to find a smaller Π-deletion set X' that is disjoint from X (line 5). This task, Disjoint Compression Task, is solved by

Algorithm: COMPRESS (G, X)
Input: An undirected graph G and a solution S
Output: A smaller solution S', if it exists, otherwise S.

```
1 for each Y ⊊ S
2     X ← S \ Y
3     G' ← G − Y
4     if G[X] ∈ Π then
5         X' ← CompressDisjoint (G', X)
6         if |X'| < |X| then return X' ∪ Y
7 return S
```

Figure 6.3: Pseudo-code of the compression routine for Π-VERTEX DELETION. Algorithm COMPRESSDISJOINT solves the DISJOINT COMPRESSION TASK.

a *disjoint compression routine* (COMPRESSDISJOINT). If such a smaller solution is found, it is returned (line 6). Otherwise, after trying all possible partitions without finding a smaller solution, we know that the solution S is optimal and return it (line 7).

The running time of the whole algorithm can be stated as follows.

Lemma 6.1. *If the* DISJOINT COMPRESSION TASK *can be solved in* t_1 *time and the property* Π *can be tested in* t_2 *time, then* Π-VERTEX DELETION *can be solved in* $O(2^k \cdot (t_1 + t_2)n)$ *time.*

Proof. The loop of algorithm ITERATE in Figure 6.1 (lines 3–8) can be executed at most n times. In each iteration, the algorithm COMPRESS in Figure 6.3 is called (line 7). After that, ITERATE aborts in line 8 if $|S| > k$. Therefore, in the next iteration of the loop, S has size at most k before adding v to it in line 6 of ITERATE. Thus, S has size at most $k + 1$ if COMPRESS is called. Therefore, COMPRESS executes its loop (lines 2–6 in Figure 6.3) at most $2^{k+1} - 1$ times. COMPRESS needs t_2 time to test whether $G[X] \in \Pi$ (line 4) and t_1 time to compress the solution X (line 5). The remaining instructions in lines 2, 3, and 6, can be executed in linear time in the worst case. Thus, each execution of COMPRESS needs $O(2^k \cdot (t_1 + t_2))$ time. In total, this sums up to a running time of $O(2^k \cdot (t_1 + t_2)n)$ for algorithm ITERATE. □

The idea to try by brute force all nontrivial partitions of a given solution S into two sets Y and X is applied for all known applications of iterative compression for Π-VERTEX DELETION problems, that is, VERTEX BIPARTIZATION [RSV04], UNDIRECTED FEEDBACK VERTEX SET [GGH+06, DFL+07, CFL+08], VERTEX COVER [Guo06], CLUSTER VERTEX DELETION (Section 6.3.1), and BOUNDED-DEGREE-1 VERTEX DELETION (Section 4.5). The main difference between these problems lies in the disjoint compression routine. The disjoint compression routines for the first two problems have exponential worst-case running time; for

Vertex Bipartization the Disjoint Compression Task is solved via a brute-force approach combined with maximum flow techniques [RSV04], and for Undirected Feedback Vertex Set it is solved with a bounded search tree approach combined with data reduction rules [CFL+08]. The disjoint compression routines for the problems Cluster Vertex Deletion and Bounded-Degree-1 Vertex Deletion run in polynomial time and are based on matching techniques, and the disjoint compression routine for Vertex Cover is a trivial algorithm applying the simple observation that a disjoint solution X' must contain all neighbors of X. From Lemma 6.1 it follows directly that for Vertex Cover, Cluster Vertex Deletion, and Bounded-Degree-1 Vertex Deletion there exists an algorithm with running time $2^k \cdot \text{poly}(n)$. The results in this chapter are driven by the question: *For which other vertex deletion problems is the* Disjoint Compression Task *polynomial-time solvable?* The computational complexity of the Disjoint Compression Task, so far, remained widely unclassified. For instance, the fixed-parameter tractability results (using iterative compression) for Vertex Bipartization [RSV04, Hüf09] or Undirected Feedback Vertex Set [GGH+06, DFL+07, CFL+08] left open whether the respective Disjoint Compression Task is NP-hard or polynomial-time solvable.

We give a complexity dichotomy for the Disjoint Compression Task of Π-Vertex Deletion for any hereditary property Π that is determined by the components, that is, that a graph G satisfies Π whenever every connected component of G satisfies Π. We can always assume that the given solution X is inclusion-minimal, that is, for every proper subset $X' \subset X$ it holds that $G - X' \notin \Pi$; if it is not inclusion-minimal, then we convert it into an inclusion-minimal solution in polynomial time as follows. Test for each $v \in X$ whether $G - (X \setminus \{v\}) \in \Pi$, and if so, set $X := X \setminus \{v\}$. The Disjoint Compression Task for Π-Vertex Deletion can therefore be stated as follows.

Disjoint Π-Vertex Deletion
Input: An undirected graph $G = (V, E)$ and a vertex subset $X \subseteq V$ such that $G[X] \in \Pi$, $G - X \in \Pi$, and X is inclusion-minimal under this property.
Question: Is there a vertex subset $X' \subseteq V$ with $|X'| < |X|$ such that $X \cap X' = \emptyset$ and $G - X' \in \Pi$?

Our original motivation for analyzing the complexity of Disjoint Π-Vertex Deletion comes from the desire to better understand the limitations of the iterative compression technique. Beyond this, Disjoint Π-Vertex Deletion also seems to be a natural and interesting problem on its own: In combinatorial optimization, one often may be confronted with finding *alternative* good solutions to already found ones. In the setting of Disjoint Π-Vertex Deletion, this is put to the extreme in the sense that we ask for solutions that are completely unrelated, that is, disjoint. For instance, this demand also naturally occurs in the context of finding quasicliques [ARS02]. The computational complexity of find-

ing alternative solutions with less strict demands has been considered in various contexts. Among others, the complexity of finding alternative solutions has been studied with respect to the Hamilton Cycle problem [Pap94, Kra99]. Finally, let us mention that there are also ties to the recent framework of reoptimization problems [BHMW08]—there, one deals with the recomputation of a solution for a locally modified input instance. In all these settings, however, one asks how the knowledge of a solution can provide structural information that helps finding an other one.

In the next section, by extending a reduction framework by Lewis and Yannakakis [LY80], we contribute a complete complexity classification (polynomial-time solvability vs. NP-completeness) of Disjoint Π-Vertex Deletion including all of the above mentioned problems.

6.3 Complexity Dichotomy for the Compression Task

This section is dedicated to the proof of the following theorem.

Theorem 6.1. *Let* Π *be any non-trivial hereditary graph property that is determined by the components and that can be tested in polynomial time, and let* $\mathcal{H}$ *be the set of forbidden induced subgraphs corresponding to* Π*.* Disjoint Π-Vertex Deletion *is NP-complete unless* $\mathcal{H}$ *contains a* P_2 *or a* P_3*, and in these cases it is polynomial-time solvable.*

Theorem 6.1 applies to many vertex deletion problems in undirected graphs, including Vertex Cover, Bounded-Degree Vertex Deletion [NRT05], Undirected Feedback Vertex Set [GGH$^+$06, DFL$^+$07, CFL$^+$08], Vertex Bipartization [RSV04], Cluster Vertex Deletion [HKMN09a], Chordal Deletion [Mar09], and Planar Deletion [MS07a]. Thus, among these problems, except for Vertex Cover ($\mathcal{H} = \{P_2\}$), Cluster Vertex Deletion ($\mathcal{H} = \{P_3\}$), and Bounded-Degree-1 Vertex Deletion ($\mathcal{H} = \{P_3, K_3\}$), all other problems have NP-complete Disjoint Π-Vertex Deletion problems.

Since the number of sets of graphs is uncountable, but the set of algorithms can be enumerated and is therefore countable, it follows that there exist Π-Vertex Deletion problems that are not in NP, that is, that the property Π cannot be tested in polynomial time. As Lewis and Yannakakis [LY80], we add the stipulation that Π can be tested in polynomial time, hence the corresponding Disjoint Π-Vertex Deletion problem is in NP, and our hardness results to come thus will show that it is NP-complete.

6.3.1 The Polynomial-Time Solvable Cases

This section covers all cases of Disjoint Π-Vertex Deletion that can be solved in polynomial time. These correspond to each graph property Π whose set

of forbidden induced subgraphs $\mathcal{H}$ contains a P_2 (a single edge) or a P_3 (a path on three vertices). Recall that we restricted our attention to hereditary graph properties that are determined by the components, that is, all graphs in H are connected.

We shortly describe the "structure" of the corresponding graph properties. If $\mathcal{H}$ contains a P_2, then this is the only forbidden induced subgraph, that is, $\mathcal{H} = \{P_2\}$, since any other connected graph contains a P_2 (recall that $\mathcal{H}$ can be assumed to be a set of minimal forbidden induced subgraphs, cf. Section 2.1). Hence, Π is the set of all graphs with no edges, that is, it is the graph property "being edgeless". If $\mathcal{H}$ contains no P_2 but a P_3, then $\mathcal{H}$ can additionally contain exact one clique, since any other connected graph on at least four vertices contains a P_3 as induced subgraph. If $\mathcal{H}$ contains only a P_3, then Π is the set of all *cluster graphs*, that is, graphs whose connected components form cliques. If $\mathcal{H}$ contains a P_3 and a clique K_t, then the corresponding graph property Π is the set of all graphs whose connected components form cliques of size at most $t-1$.

In the following, we name these graph properties according to the following definition.

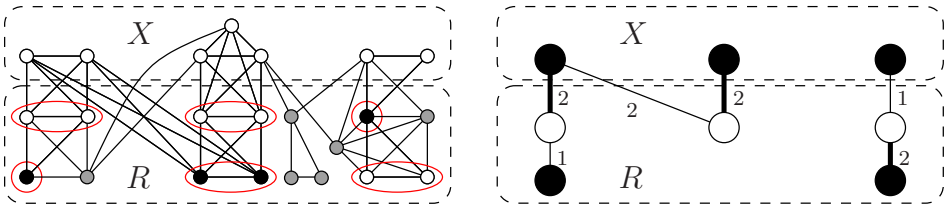
Definition 6.1. *Let Π_s, for $s \geq 1$, be the graph property that consists of all graphs whose connected components are cliques of at most s vertices. Furthermore, let Π_∞ be the graph property that consists of all graphs whose connected components of G are cliques (of arbitrary size).*

For instance, Π_1, Π_2, and Π_∞ are the properties "being edgeless", "being a graph of maximum degree one", and "being a cluster graph", respectively. As described above, the corresponding sets of minimal forbidden induced subgraphs consist of: P_2 (Π_1), P_3 and K_3 (Π_2), and P_3 (Π_∞). In general, the set of forbidden induced subgraphs of Π_s for $s \geq 2$ contains P_3 and K_{s+1}. Summarizing, for each property Π_s, $s \geq 1$, and Π_∞, the corresponding set of forbidden induced subgraphs contains a star with at most two leaves (in other words, a P_2 or a P_3), and these are the only properties whose sets of forbidden induced subgraphs contain a star with at most two leaves.

Theorem 6.2. Disjoint Π-Vertex Deletion *can be solved in polynomial time if $\Pi = \Pi_s$, for some $s \geq 1$, or if $\Pi = \Pi_\infty$.*

For property Π_1, the disjoint solution X' must contain every endpoint of each edge that has one endpoint in the given solution X and the other endpoint in $V \setminus X$. Hence, the input is a yes-instance if and only if X forms an independent set and $|N_G(X)| < |X|$. This condition can be tested in linear time by scanning through all edges incident to each vertex in X. Recall that in this theses n and m are the number of vertex and edges in the graph, respectively.

Lemma 6.2. Disjoint Π_1-Vertex Deletion *can be solved in $O(n+m)$ time,*



(a) Disjoint Π_∞-Vertex Deletion instance (b) The assignment problem

Figure 6.4: (a) Data reduction in the disjoint compression routine. The gray vertices in the input instance are deleted by the data reduction rules. The black vertices correspond to a minimum solution X_2' (determined by a solution for the assignment problem, see below). Each circled group of vertices corresponds to an edge in the assignment problem (b). If a circled group of vertices is in X or if no vertex in this group has a neighbor in X, then the corresponding vertex is black, otherwise, it is white. The bold edges show the maximum matching that corresponds to the minimum solution X_2'.

The proof of the next lemma exhibits the main technique, which we will adapt for the proof of Lemma 6.4, that shows the polynomial-time solvability for the remaining properties Π_s for $s \geq 2$.

Lemma 6.3. Disjoint Π_∞-Vertex Deletion *can be solved in* $O(m\sqrt{n} \cdot \log n)$ *time.*

Proof. Recall that the set of minimal forbidden induced subgraphs corresponding to Π_∞ only contains one element, namely a P_3. We describe an algorithm that solves Disjoint Π_∞-Vertex Deletion. In some sense, it is a generalized version of the algorithm for the disjoint compression task for Bounded-Degree-1 Vertex Deletion (see Section 4.5) as it also uses matching techniques. However, in the case of CVD Compression, the resulting matching instance is more complicated and cannot be solved in linear time. An example for a Disjoint Π_∞-Vertex Deletion instance is shown in Figure 6.4a. The algorithm begins by computing all vertices that necessarily have to be in X'; if there exists an induced P_3 in G such that exactly one vertex v of that P_3 is in $R := V \setminus X$ (thus, the remaining two vertices of the P_3 are in X), then we call v a *necessary vertex*. Obviously, all necessary vertices have to be in X'. Thus, a first data reduction rule computes the set of necessary vertices, adds them to an initially empty set X_1', and deletes the necessary vertices from the graph G and from the set R. After that, a second data reduction rule deletes connected components that are cliques from G, from R, and from X; the soundness of this rule is obvious. In the following, let (G, X) be the remaining instance after deleting all necessary vertices and isolated cliques. The set X_1' contains all necessary vertices, thus it remains to compute an optimal solution X_2' for (G, X); $X' := X_1' \cup X_2'$ is then a minimum-size solution for our input instance.

The reduced instance (G, k) is much simplified: In each clique of $G[R]$, we can divide the vertices into equivalence classes according to their neighborhood in X; each class then contains either vertices adjacent to all vertices of a particular clique in $G[X]$, or the vertices adjacent to no vertex in X (see Figure 6.4a), otherwise, there would be a necessary vertex and the first data reduction rule above would apply. This classification is useful because of the following:

Claim. *If there exists a solution for* Disjoint Π_∞-Vertex Deletion*, then in the cluster graph resulting by this solution, each clique in* $G[R]$ *consists of vertices of at most one equivalence class.*

Proof of Claim. Clearly, inside a clique, it is never useful to delete only some, but not all vertices of an equivalence class, since if that led to a solution, we could always re-add the deleted vertices without introducing new induced P_3's. Further, assume that for a clique C in $G[R]$ the vertices of two equivalence classes are present. Let $u \in C$ and $v \in C$ be a vertex from each equivalence class, respectively. Since u and v are in different equivalence classes, they must have a different neighborhood with respect to the cliques in $G[X]$. Assume without loss of generality that v is adjacent to all vertices of a clique C' in $G[X]$. Since u is in an other equivalence class than v, u is not adjacent to any vertex of C'. Let $w \in C'$. The path uvw forms an induced P_3, contradicting our assumption and showing the claim.

Due to this claim, the remaining task for solving Disjoint Π_∞-Vertex Deletion is to assign each clique in $G[R]$ to one of its equivalence classes (corresponding to the preservation of this class, and the deletion of all vertices from the other classes within the clique) or to do nothing (corresponding to the complete deletion of the clique). However, we cannot do this independently for each clique; we must not choose two classes from different cliques in $G[R]$ such that these two classes are adjacent to the same clique in $G[X]$ since that would create an induced P_3. This assignment problem can be modeled as a weighted bipartite matching problem in an auxiliary graph H, where each edge corresponds to a possible choice. The graph H is constructed as follows (see Figure 6.4b):

1. Add a vertex for every clique in $G[R]$ (white vertices).

2. Add a vertex for every clique in $G[X]$ (black vertices in X).

3. For a clique C_X in $G[X]$ and a clique C_R in $G[R]$, add an edge between the vertex for C_X and the vertex for C_R if there is an equivalence class in C_R containing a vertex adjacent to a vertex in C_X. This edge corresponds to choosing this class for C_R and one assigns the number of vertices in this class as its weight.

4. Add a vertex for each class in a clique C_R that is not adjacent to the cliques in $G[X]$ (black vertices outside X), and connect it to the vertex

representing C_R. Again, this edge corresponds to choosing this class for C_R and is weighted with the number of vertices in this class.

Since we only added edges between black and white vertices, H is bipartite. The task is now to find a *maximum-weight bipartite matching*, that is, a set of edges of maximum weight where no two edges have an endpoint in common. To solve this matching instance, we can use an algorithm for integer-weighted matching [GT89] with a maximum weight of n (since a class can contain at most n vertices), yielding a running time of $O(m\sqrt{n}\log n)$. The set X'_2 can be directly constructed from a maximum matching; it contains all vertices in equivalence classes in $G[R]$ that correspond to edges not chosen by the matching in H (see Figure 6.4). If we apply the data reduction rules in their given order, we can execute them in $O(m)$ time. Obviously, the input instance is a yes-instance if and only if $|X'_1| + |X'_2| < |X|$. Thus, we can solve Disjoint Π_∞-Vertex Deletion in $O(m\sqrt{n}\log n)$ time. □

With some tricks based on problem kernelization, which will not be explained here, the running time can be further improved to $O(2^k k^6 \log k + nm)$ [HKMN09a]. This iterative compression approach combined with matching techniques also works for the vertex-weighted version of Cluster Vertex Deletion, yielding an algorithm with a running time of $O(2^k k^9 + nm)$ [HKMN09a].

It remains to show the polynomial-time solvability for the remaining properties Π_s. The technique is similar.

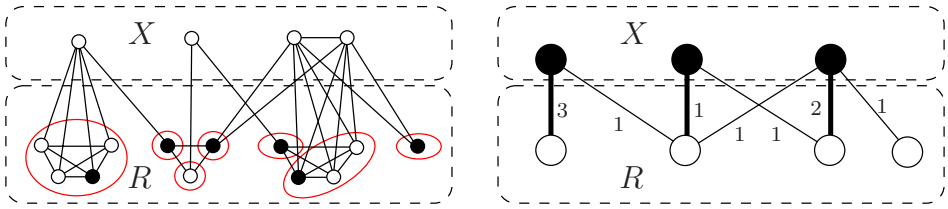
Note that Disjoint Π_2-Vertex Deletion is equivalent to the disjoint compression task of Bounded-Degree-1 Vertex Deletion, which can be solved in linear time (Theorem 4.5 in Section 4.5).

Lemma 6.4. *For each $s \geq 2$,* Disjoint Π_s-Vertex Deletion *can be solved in $O(m\sqrt{n}\log n)$ time.*

Proof. Let (G, X) be the input instance for Disjoint Π_s-Vertex Deletion. Recall that the set of minimal forbidden induced subgraphs corresponding to Π_s is a P_3 and a clique of $s+1$ vertices; hence, every connected component of $G - X$ is a clique of at most s vertices.

We describe an algorithm that finds a minimum-size vertex set X' such that $X \cap X' = \emptyset$ and such that $G - X' \in \Pi_s$, or returns "no-instance". This algorithm is similar to the one for Disjoint Π_∞-Vertex Deletion in the proof of Lemma 6.3, but additionally takes into account the forbidden clique of $s+1$ vertices.

The algorithm starts by computing all vertices that necessarily have to be in X': if there exists a forbidden induced subgraph F in G such that exactly one vertex v of F is in $R := V \setminus X$ (thus, all remaining vertices of F are in X), then we call v a *necessary vertex*. Obviously, all necessary vertices have to be in X'. Thus, a first data reduction rule computes the set of necessary vertices, adds them to an initially empty set X'_1, and deletes the necessary vertices from G. This can be easily accomplished by first finding and deleting all necessary vertices due to



(a) Disjoint Π_4-Vertex Deletion instance (b) The assignment problem

Figure 6.5: (a) Instance (G, X), preprocessed as described in the proof of Lemma 6.4, for $s = 4$. The black vertices are in a minimum solution X_2'. Each group of encircled vertices corresponds to an edge in the assignment problem (b). The edges in the assignment problem are weighted with the number of vertices that do not have to be taken into X_2' if the corresponding edge is chosen to be in the matching. The bold edges show the maximum matching that corresponds to the minimum solution X_2'.

the forbidden P_3 (as in the proof of Lemma 6.3), and then finding and deleting all necessary vertices due to the forbidden clique of $s + 1$ vertices (clearly, the neighborhood of every s-vertex clique in $G[X]$ is a set of necessary vertices due to the forbidden clique). Then, consider the connected components C in the graph reduced by the first rule. For each C that is a clique, a second data reduction rule adds $|V(C)| - s$ arbitrary vertices from $V(C) \cap R$ to X_1' if $|V(C)| > s$ (there are always sufficiently many vertices to choose from, since $G[X]$ only contains cliques of size at most s), and deletes C from the graph. This reduction rule is correct, because for a connected component that is a clique of more than s vertices it does not matter which vertices are deleted, as there is no connection to the rest of the graph.

In the following, let (G, X) be the remaining instance after exhaustively applying the two data reduction rules. As in the proof of Lemma 6.3, a minimum-size set X_2', $X_2' \cap X = \emptyset$, containing a vertex of every induced path on three vertices and every clique of $s+1$ vertices, can be obtained with matching techniques. The main difference is that if a clique in $G[R]$ is present in the cluster graph $G - X'$, then it is not necessarily the case that all vertices of that clique are present due to the size constraint s. This size constraint, however, can be encoded in the edge weights of the corresponding assignment problem. The construction of H is the same as in the proof of Lemma 6.3 except for the weight of an edge between the vertex for a clique C_X in $G[X]$and the vertex for a clique C_R in $G - X$ (assuming that there is an equivalence class in C_R that contains a vertex adjacent to C_X): the weight of the edge is set to $\min\{s - |V(C_X)|, t\}$, where t is the number of vertices in the corresponding equivalence class in C_R. If the edge is in a maximum matching of H, and if $t \leq s - |V(C_X)|$, then the argument is as in the proof of Lemma 6.3 and no vertex of the corresponding class in C_R is in X_2'; however, if $t > s - |V(C_X)|$, then this still means that the corresponding class

of C_R is chosen, but since together with the vertices in C_X there are more than s vertices, one has to add all but $s - |V(C_X)|$ vertices of this class to X_2'. Consider Figure 6.5 for an example of such an instance (G, X) and the corresponding assignment problem.

The data reduction rules can be performed in $O(m)$ time, and the matching algorithm needs $O(m\sqrt{n}\log n)$ time (see proof of Lemma 6.3). □

6.3.2 NP-Hardness Framework and Simple Proofs

Lewis and Yannakakis [LY80] showed that Π-Vertex Deletion for any non-trivial hereditary property Π is NP-complete. Due to the similarity of Π-Vertex Deletion to Disjoint Π-Vertex Deletion, in some simple cases we can adapt the framework from [LY80].[4] This section is mainly devoted to this framework and how it can be modified to partially address the complexity of Disjoint Π-Vertex Deletion.

There are cases, however, where our adaption fails; this happens when there is a star with at least three leaves among the family $\mathcal{H}$ of minimal forbidden induced subgraphs corresponding to Π. For this case, we have to devise other NP-hardness proofs (if there is a star with at most two leaves, then the problem is polynomial-time solvable). Summarizing, we have to distinguish the following three cases (recall that each graph in $\mathcal{H}$ is connected, because Π is determined by the components, cf. Section 2.1):

1. $\mathcal{H}$ does not contain a star (NP-hard, this section, Theorem 6.3), and
2. $\mathcal{H}$ contains a star with at least three leaves (NP-hard, Section 6.3.3, Theorem 6.4), and
3. $\mathcal{H}$ contains a star with at most two leaves (that is, a P_2 or a P_3; polynomial-time solvable, Section 6.3.1, Theorem 6.2).

The main result of this section covers all cases that can be proven by adapting the framework of Yannakakis.

Theorem 6.3. *Let* Π *be a non-trivial hereditary property that is determined by the components and let* $\mathcal{H}$ *be the corresponding set of all forbidden induced subgraphs. If* $\mathcal{H}$ *contains no star, then* Disjoint Π-Vertex Deletion *is NP-hard.*

The Framework of Yannakakis

In the following, we briefly describe the reduction by Yannakakis [LY80], showing that any vertex deletion problem for a non-trivial hereditary graph property is

[4] As made explicit in Lewis and Yannakakis' paper [LY80], the parts of it we are referring to in our work have been contributed by Yannakakis. That is why we refer to it in the following as "the framework of Yannakakis".

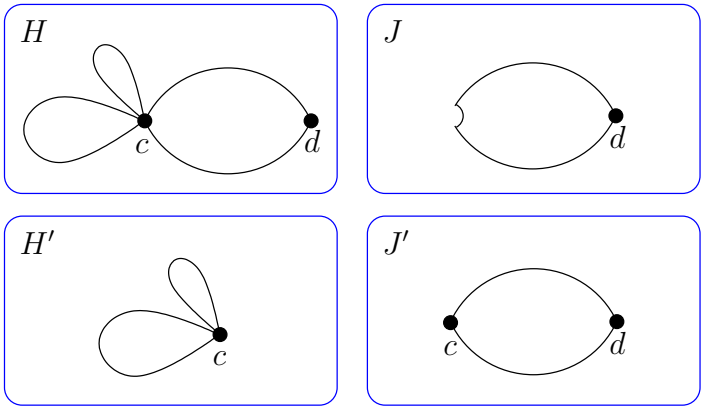


Figure 6.6: Connected graph H with cut-vertex c and some vertex d in a largest connected component J of $H - c$. Moreover, the graphs $H' = H - V(J)$ and $J' = H[V(J) \cup \{c\}]$ are illustrated.

NP-hard. Since the hereditary graph properties considered in this paper are assumed to be determined by the components, we present a variant that is restricted to such properties, that is, the forbidden induced subgraphs are connected.

Preliminaries. Let $\mathcal{H}$ be the set of minimal forbidden induced subgraphs that correspond to the non-trivial hereditary property Π that is determined by its components. In the following, we call a vertex subset X such that $G - X \in \Pi$ a *$\mathcal{H}$-obstruction set* in G (since it obstructs every forbidden induced subgraph in $\mathcal{H}$). An important concept for the framework is the notion of α-sequences [LY80].

Definition 6.2 (α-sequence)**.** *For a connected graph $H \in \mathcal{H}$, if H is 1-connected, then take a cut-vertex c; otherwise, then let c be an arbitrary vertex (in this case, $H - c$ has just one connected component). Sorting the connected components of $H - c$ decreasingly with respect to their sizes gives a sequence $\alpha = (n_1, \ldots, n_i)$, where $n_1 \geq \ldots \geq n_i$. The sequence depends on the choice of c. The* α-sequence *of H, $\alpha(H)$, is a sequence which is lexicographically smallest among all such sequences α.*

Let $H \in \mathcal{H}$ be a graph with the lexicographically smallest α-sequence among all graphs in $\mathcal{H}$. Note that every proper induced subgraph of H has a lexicographically smaller α-sequence than H. Since Π is satisfied by all edge-less graphs (Π is determined by the components, thus it contains all edge-less graphs), the connected graph H must contain at least two vertices, thus a largest component J of $H - c$ contains at least one vertex. Let d be an arbitrary vertex in J, and let H' be the graph resulting by removing all vertices in J from H, and let J' be the subgraph of H induced by $V(J) \cup \{c\}$. See Figure 6.6 for an example.

Reduction. The reduction by Yannakakis [LY80] from the NP-complete Vertex Cover problem works as follows. Let G be an instance of Vertex Cover.

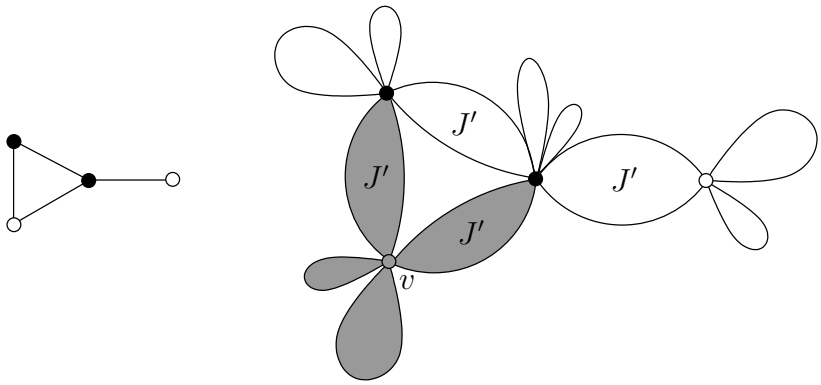


Figure 6.7: Example for a reduction from Vertex Cover. Left: Vertex Cover instance with a vertex cover A (black vertices). Right: Corresponding Π-Vertex Deletion instance G' constructed by the framework of Yannakakis, together with a solution X (black vertices) corresponding to the vertex cover A. The gray area marks a connected component in $G' - X$ corresponding to case (2) in the correctness proof with cut-vertex v.

For every vertex v in G create a copy of H' and identify c and v. Replace every edge $\{u, v\}$ in G by a copy of J', identifying c with u and d with v. Let G' be the resulting graph. See Figure 6.7 for an example of the reduction.

Correctness. The graph G has a size-k vertex cover if and only if G' has a size-k vertex set that obstructs the set of forbidden induced subgraphs $\mathcal{H}$ in G':

($\Rightarrow$) If A is a vertex cover of G, then $X' := A$ also obstructs all graphs in $\mathcal{H}$: Every connected component of $G' - X'$ is either

1. a connected component of a copy of $H' - c$ or
2. a copy of H' together with several copies of J', each with c or d deleted.

Let C be a connected component of $G' - X'$. In case (1), $\alpha(H' - c)$ is lexicographically smaller than $\alpha(H)$ since $H' - c$ is a subgraph of H. In case (2), the copy of H' and the copies of J' intersect exactly in one vertex v of $V(G)$. Hence, v is a cut-vertex and the components of $C - v$ can be divided into a copy of $H' - c$ and several copies of J with one vertex deleted. Since the latter type of components has less than $|V(J)|$ vertices, the cut-vertex v gives an α-sequence for C which is lexicographically smaller than the α-sequence of H (see also Figure 6.7). As a consequence, the connected components in $G' - X'$ have a smaller α-sequence than H, and because H is a forbidden induced subgraph with lexicographically smallest α-sequence, these connected components do not contain forbidden induced subgraphs.

($\Leftarrow$) If X' is a solution for Π-Vertex Deletion, then one can determine a vertex cover A for G as follows: for each $w \in X'$, if w is in a copy of H'

(possibly $w \in V(G)$), then add vertex c of that copy of H' to A, and if w is in a copy of J' (where $w \notin V(G)$), then add vertex c of that copy of J' to A. Obviously, $|A| \leq |X'|$. Suppose that there exists an edge $\{u, v\}$ in $G - A$. Then, by construction of A, X' neither contains any vertex from the two copies of H' corresponding to the vertices u and v nor from the copy of J' that replaced the edge $\{u, v\}$ in the construction of G'. Hence $G' - X'$ contains a copy of H, a contradiction. Therefore, A is a vertex cover for G.

Limitations. In some cases, a very similar reduction principle can be applied for Disjoint Π-Vertex Deletion. The main difficulty in the case of Disjoint Π-Vertex Deletion compared to Π-Vertex Deletion is that one has to construct an old solution X such that X does not prevent the "equivalence argument" of the reduction. In other words, one has to construct an $\mathcal{H}$-obstruction set X in G' with the restriction that X does not contain any vertex from $V(G)$. Then, in principle, we can use the same arguments as above. However, for some cases this approach fails; for instance, if J' is a clique and some graph of $\mathcal{H}$ is contained in G, then this forbidden induced subgraph, which also exists in G', can only be obstructed by vertices from $V(G)$. For example, this happens when Π is the property "being cycle-free" (Feedback Vertex Set): $\mathcal{H}$ contains all cycles, and the graph H with the smallest α-sequence is the K_3. Then, cycles in G exist also in G' and can only be obstructed by vertices from $V(G)$. One can deal with this situation by reducing from K_3-free graphs, and using the graph with the smallest α-sequence among all K_3-free graphs in $\mathcal{H}$, as shown in the proof of Lemma 6.7. The same type of problem, however, also occurs if H is a star. In this case, each connected component of $H - c$ is an isolated vertex. Thus, the vertex d has to be one of these vertices, and G and therefore G' might contain a forbidden induced subgraph with lexicographically higher α-sequence than H. This induced subgraph cannot be obstructed by a set X that is not allowed to contain any vertex from $V(G)$. In this case, the framework of Yannakakis seems not to be suitable and we use different arguments (Section 6.3.3).

Proofs Based on the Reduction Framework of Yannakakis

First, we introduce a new variant of Disjoint Π-Vertex Deletion, where the size of the new solution is given as a parameter, and show how it can be reduced to Disjoint Π-Vertex Deletion. In all proofs that follow, we show the NP-hardness of that variant, because all hardness proofs need a *size gadget* that is employed in the reduction from the variant to Disjoint Π-Vertex Deletion.

> Size Disjoint Π-Vertex Deletion
> **Input:** An undirected graph $G = (V, E)$, a parameter k, and a vertex subset $X \subseteq V$ such that $G[X] \in \Pi$, $G - X \in \Pi$, and X is inclusion-minimal under this property.
> **Question:** Is there a vertex subset $X' \subseteq V$ with $|X'| \leq k$ such that $X \cap X' = \emptyset$ and $G - X' \in \Pi$?

Lemma 6.5. Size Disjoint Π-Vertex Deletion *can be reduced to* Disjoint Π-Vertex Deletion *in polynomial time.*

Proof. Let (G, X, k) be an instance of Size Disjoint Π-Vertex Deletion. We construct an instance $(\hat{G}, \hat{X})$ of Disjoint Π-Vertex Deletion as follows (recall that Disjoint Π-Vertex Deletion asks for a solution $\hat{X}'$ such that $|\hat{X}'| < |\hat{X}|$). First, suppose that $k = |X| - 1$. Then, obviously, (G, X, k) is a yes-instance for Size Disjoint Π-Vertex Deletion if and only if $(\hat{G}, \hat{X}) := (G, X)$ is a yes-instance for Disjoint Π-Vertex Deletion. It remains to deal with the cases $k < |X| - 1$ and $k > |X| - 1$, where we employ a padding trick using a *size gadget*. The basic idea is to enforce that, in the constructed graph G' containing the size gadget, a certain number of vertices of the new solution has to be in the size gadget, such that there are exactly k vertices left to obstruct all forbidden induced subgraphs in G.

Size gadget for $k < |X| - 1$: In this case, informally speaking, we have to force that only k vertices out of the $|X| - 1$ available vertices can be used to obstruct all forbidden induced subgraphs. Let H, c, J, J', and d be defined as in the reduction scheme (see Figure 6.6). We create a new graph $\hat{G}$ by using a copy of G and adding a padding gadget C constructed as follows. Add a new vertex w and $|X| - k$ copies of H, identify the vertex d of each newly added copy of H with w, and let $\hat{X} := X \cup \{w\}$. The gadget C is obviously connected and w is a cut-vertex in C. The vertex w obstructs all forbidden induced subgraphs in C, because deleting w (and, thus, d) from each copy of H in C leaves a graph with lexicographically smaller α-sequence (witnessed by c in each copy of H). Hence, $\hat{X}$ is a minimal $\mathcal{H}$-obstruction set for $\hat{G}$.

An $\mathcal{H}$-obstruction set $\hat{X}'$ for $\hat{G}$ with $\hat{X}' \cap \hat{X} = \emptyset$ must contain at least one vertex in each copy of H in C, thus $\hat{X}'$ must contain at least $|X| - k$ vertices of C; putting into $\hat{X}'$ the vertex c of each copy of H in C obstructs every forbidden induced subgraph in H: every connected component of $C - \hat{X}'$ either is a connected component of a copy of $H - c$ or consists of $|X| - k$ copies of J that pairwise overlap in the vertex w. In the latter case, w is a cut-vertex witnessing that each remaining connected component has size smaller than J, yielding a lexicographically smaller α-sequence. This shows that $\hat{X}'$, in order to obstruct all forbidden induced subgraphs in C, needs to contain at least $|X| - k$ vertices of C. Recall that one demands that $|\hat{X}'| < |\hat{X}|$. Since $\hat{X} = X \cup \{w\}$, there remain at most $|\hat{X}| - |X| + k - 1 = k$ vertices to obstruct all forbidden induced subgraphs in $G = \hat{G} - V(C)$. Hence, (G, X, k) is a yes-instance for Size Disjoint Π-Vertex Deletion if and only if $(\hat{G}, \hat{X})$ is a yes-instance for Disjoint Π-Vertex Deletion.

Size gadget for $k > |X| - 1$: In this case, we construct $\hat{G}$ in the same manner using a gadget C with $k - |X| + 2$ copies of H overlapping in vertex w and let $\hat{X}$ be the union of X and the vertex c of each copy of H. Then, $|\hat{X}| = k + 2$. A new solution $\hat{X}'$ of size at most $|\hat{X}| - 1 = k + 1$ with $\hat{X}' \cap \hat{X} = \emptyset$ for $\hat{G}$ can obstruct all forbidden induced subgraphs in C with the vertex w, and there are k vertices left

to obstruct all forbidden induced subgraphs in $G = \hat{G} - V(C)$. Hence, (G, X, k) is a yes-instance for Size Disjoint Π-Vertex Deletion if and only if $(\hat{G}, \hat{X})$ is a yes-instance for Disjoint Π-Vertex Deletion.

Obviously, in both cases the size gadget C can be constructed in polynomial time. □

Recall that, for the following two proofs, we assume that the set $\mathcal{H}$ of forbidden induced subgraphs corresponding to Π contains no star. We have to distinguish between the cases that

1. all forbidden induced subgraphs in $\mathcal{H}$ contain a K_3 (see Lemma 6.6), and that
2. not all forbidden induced subgraphs in $\mathcal{H}$ contain a K_3 (see Lemma 6.7).

Lemma 6.6. *If the set $\mathcal{H}$ of forbidden induced subgraphs corresponding to* Π *only consists of graphs that contain a K_3, then* Disjoint Π-Vertex Deletion *is NP-hard.*

Proof. The proof is by reduction from the NP-complete Vertex Cover problem on K_3-free graphs [GJ79] to Size Disjoint Π-Vertex Deletion. Let (G, k) be an instance of Vertex Cover, where G is K_3-free. First, construct a graph G' using the reduction scheme by Yannakakis. Greedily compute a minimal $\mathcal{H}$-obstruction set X for G' such that $X \cap V(G) = \emptyset$. Such a set X always exists, since G is K_3-free and, therefore, does not contain any forbidden induced subgraph. By these arguments and the reduction scheme, G has a size-k vertex cover if and only if (G', X, k) is a yes-instance for Size Disjoint Π-Vertex Deletion. The NP-hardness of Disjoint Π-Vertex Deletion then follows from Lemma 6.5. □

In the following, assume that not all forbidden induced subgraphs contain a K_3.

Lemma 6.7. *If the set $\mathcal{H}$ of forbidden induced subgraphs corresponding to* Π *contains no stars, but other graphs that do not contain a K_3, then* Disjoint Π-Vertex Deletion *is NP-hard.*

Proof. The reduction from the NP-complete Vertex Cover on K_3-free graphs is very similar to the one for Lemma 6.6. The difference is that G might now contain a forbidden subgraph in $\mathcal{H}$, and we have to show that we can greedily compute a minimal $\mathcal{H}$-obstruction set X for G' such that $X \cap V(G) = \emptyset$ (as in the proof of Lemma 6.6). To this end, we first set the encoding forbidden subgraph H used in the reduction scheme by Yannakakis equal to a K_3-free subgraph in $\mathcal{H}$ which has the lexicographically smallest α-sequence among all K_3-free graphs in $\mathcal{H}$. Second, by setting H in this way, a largest connected component in $H - c$ contains at least one edge, due to the fact that H is not a star. Moreover, since H does not contain K_3, at most one endpoint of this edge is adjacent to c. Then,

we select an endpoint of this edge that is not adjacent to c as the vertex d used in the construction of G'. Now, we can observe that in the resulting G' the vertices in $V(G)$ induce an independent set. Therefore, removing all vertices $V(G')\setminus V(G)$ gives an $\mathcal{H}$-obstruction set for G' and we can easily compute an inclusion-minimal solution X for G' with $X \cap V(G) = \emptyset$. Since the graphs G and H are K_3-free and so is G', forbidden induced subgraphs with a smaller α-sequence than H, that contain a K_3, do not have to be considered, and the correctness of the reduction for this case follows from the same arguments as in the proof of Lemma 6.6. □

6.3.3 Refined Reduction Strategies

Here, we present NP-hardness proofs for the cases where we have a star with at least three leaves as a forbidden subgraph. The main result of this section is as follows.

Theorem 6.4. *Let Π be a non-trivial hereditary graph property that is determined by the components and let $\mathcal{H}$ be the corresponding set of all minimal forbidden induced subgraphs. If $\mathcal{H}$ contains a star with at least three leaves, then* Disjoint Π-Vertex Deletion *is NP-hard.*

Note that a star has a smaller α-sequence than any other forbidden induced subgraph that is not a star, and there is only one star in $\mathcal{H}$, since the graphs in $\mathcal{H}$ are inclusion-minimal. Therefore, if $\mathcal{H}$ contains a star, then the graph with smallest α-sequence is necessarily the star in $\mathcal{H}$. Let H be the star in $\mathcal{H}$.

The proof of Theorem 6.4 is based on the following case distinctions.

1. H is a star with at least four leaves (Lemma 6.8).
2. H is a star with three leaves.
   (a) $\mathcal{H}$ contains a P_4 (Lemma 6.9).
   (b) $\mathcal{H}$ does not contain a P_4 (Lemma 6.10).

Lemma 6.8. *If the set $\mathcal{H}$ of forbidden induced subgraphs corresponding to property Π contains a star H with at least four leaves, then* Disjoint Π-Vertex Deletion *is NP-hard.*

Proof. The proof is by reduction from the NP-complete Vertex Cover on graphs of maximum degree three [GJ79] to Size Disjoint Π-Vertex Deletion. Let (G, k) be a corresponding input instance of Vertex Cover. Let $l \geq 4$ be the number of leaves of H. An example of the following construction is given in Figure 6.8. Starting with an empty graph G' and an empty solution set X, for each vertex v in G, create a copy H_v of a star with $l-1$ leaves (vertex gadget), identify its center vertex with v, and add any $\deg_G(v)$ of H_v's leaves to X. For each edge $\{u, v\}$ in G, create a copy $H_{\{u,v\}}$ of H (edge gadget), add $H_{\{u,v\}}$'s center

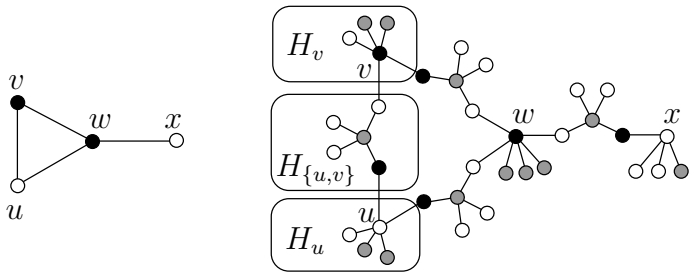


Figure 6.8: Example for the reduction in the proof of Lemma 6.8 if H is a star with four leaves. Left: Vertex Cover instance with a vertex cover C (black vertices). Right: Corresponding Size Disjoint Π-Vertex Deletion instance with the given solution X (gray vertices) and a solution X' corresponding to the vertex cover C (black vertices). For illustration, the gadgets H_u, H_v, and $H_{\{u,v\}}$ are labeled.

vertex to X, select two arbitrary leaves u', v' of $H_{\{u,v\}}$ and insert the edges $\{u, u'\}$ and $\{v, v'\}$.

Obviously, the graph $G' - X$ only contains connected components that are either isomorphic to a star with $l-1$ leaves or isolated vertices. An isolated vertex as well as a star with $l-1$ leaves both have lexicographically smaller α-sequences than H. Hence, since the star H with at least four leaves has a lexicographically smallest α-sequence among all graphs in $\mathcal{H}$, $G'-X \in \Pi$. Moreover, X is minimal, because $G-(X \setminus \{v\})$ does contain a star with l leaves for any $v \in X$. It remains to show that there is a size-k vertex cover for G if and only if there is a vertex set X', $X' \cap X = \emptyset$, of size $k' := k + |E(G)|$, obstructing all forbidden induced subgraphs in G'. In other words, (G, k) is a yes-instance for Vertex Cover if and only if (G', X, k') is a yes-instance for Size Disjoint Π-Vertex Deletion, which shows that Size Disjoint Π-Vertex Deletion is NP-hard. The NP-hardness of Disjoint Π-Vertex Deletion then follows from Lemma 6.5.

($\Rightarrow$) Let C be a size-k vertex cover for G. The set X' is constructed as follows. Beginning with $X' := C$, for each copy $H_{\{u,v\}}$ with the two leaves u', v' (see construction), if $u \in C$ and $v \notin C$, then add v' to X', if $u \notin C$ and $v \in C$, then add u' to X', and if $u \in C$ and $v \in C$, then add either u' or v' to X'. Clearly, $|X'| = k + |E(G)|$, since X' contains a vertex for each edge in G and k vertices from the size-k vertex cover C (also see Figure 6.8). The connected components in $G' - X'$ are either isolated vertices or stars with $l-1$ leaves: the leaves of the components H_v are isolated vertices in $G' - X'$ if $v \in X'$, and if $v \notin X'$, then its neighbors on the adjacent edge-gadgets are in X' by construction of X'. Hence, the vertex-gadget H_v, a star with $l-1$ leaves, forms a connected component in $G'-X'$. Concerning an edge-gadget $H_{\{u,v\}}$, we observe that exactly one of the two vertices u', v' is in X'. Without loss of generality assume that $v' \in X'$ and $u' \notin X'$. Then, by the construction of X', $u \in X'$. Thus, $H_{\{u,v\}} - v'$,

a star with $l-1$ leaves, is a connected component in $G' - X'$. Since H is the forbidden subgraph with the lexicographically smallest α-sequence, X' obstructs all forbidden induced subgraphs in G'.

($\Leftarrow$) Let X' be a size-$(k + |E(G)|)$ vertex set that obstructs every forbidden induced subgraph in G'. We may assume that X' does not contain any degree-one vertex of G' (since a degree-one vertex in X' of a vertex gadget could be simply replaced by its neighbor, and a degree-one vertex in X' of an edge gadget $H_{\{u,v\}}$ could be simply replaced by either u' or v'). Observe that for each edge-gadget $H_{\{u,v\}}$ at least one of u', v' must be in X', since $H_{\{u,v\}}$ is a forbidden induced subgraph. Hence, X' contains at least $|E(G)|$ vertices of the edge gadgets. Let $\{u, v\}$ be an edge in G. We distinguish two cases:

1. If only one of u', v' of $H_{\{u,v\}}$ is in X', then u or v is in X': assume without loss of generality that $u' \in X'$ and $v' \notin X'$. Then, v' together with the vertices of H_v induce a star with l leaves (which is forbidden) in G', and since we assumed that the leaves of H_v are not in X', $v \in X'$.

2. If both u' and v' are in X', and $u, v \notin X'$, then we can simply remove u' from X' and add u instead. After that, X' still obstructs all forbidden induced subgraphs, and case (1) applies.

Hence, for each edge $\{u, v\}$ in G at least one of its endpoints is in X'. In other words, $X' \cap V(G)$ is a vertex cover for G. Since X' contains at least $|E(G)|$ vertices of the edge gadgets, $X' \cap V(G)$ has size at most k. □

Next, we show the NP-hardness of the case that the forbidden subgraph with the lexicographically smallest α-sequence is a star with three leaves. In this case, a reduction from Vertex Cover seems less promising, since the Vertex Cover instance we reduce from contains vertices of degree three and therefore copies of the forbidden induced star with three leaves, which would have to be obstructed by the solution X in the reduction. This makes is difficult to translate a solution for Size Disjoint Π-Vertex Deletion back to a vertex cover in the Vertex Cover instance.

We reduce from 3-CNF-SAT. Our proofs rely heavily on the simple structure of a star. First, we consider the case that the path on four vertices is also forbidden.

Lemma 6.9. *If the set $\mathcal{H}$ of forbidden induced subgraphs corresponding to property Π contains a star H with three leaves and $\mathcal{H}$ also contains the path on four vertices, then* Disjoint Π-Vertex Deletion *is NP-hard.*

Proof. The proof is by reduction from 3-CNF-SAT to Size Disjoint Π-Vertex Deletion. We assume without loss of generality that each variable appears in each clause at most once. Let $F = c_1 \wedge \cdots \wedge c_q$ be a 3-CNF formula over a variable set $Y = \{y_1, \ldots, y_p\}$. We denote the kth literal in clause c_j by l_j^k, for $1 \leq k \leq 3$. An example of the following construction is given in Figure 6.9. Starting with

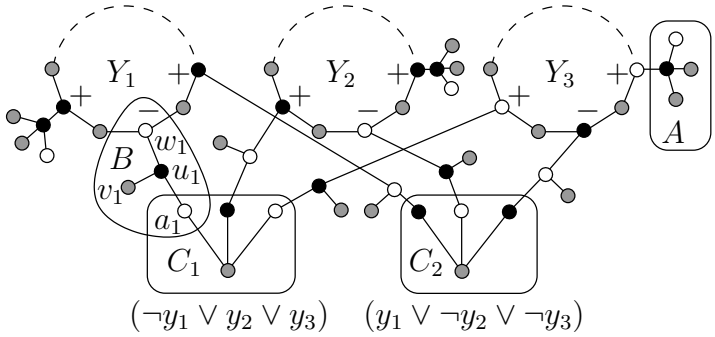


Figure 6.9: Example for the reduction in the proof of Lemma 6.9 for the 3-CNF-SAT formula $(\neg y_1 \vee y_2 \vee y_3) \wedge (y_1 \vee \neg y_2 \vee \neg y_3)$. For illustration, one minimality gadget is labeled with A and one connection gadget is labeled with B. The vertices of the connection gadget B are named according to the definitions of a_k, u_k, v_k, and w_k in the proof of Lemma 6.9 for $k = 1$. The vertices in the given solution X are gray, the vertices in the disjoint solution X', corresponding to the satisfying truth assignment $y_1 = \text{true}$, $y_2 = \text{true}$, $y_3 = \text{false}$, are black.

an empty graph G and $X := \emptyset$, construct an instance (G, X) for Disjoint Π-Vertex Deletion as follows. For each variable y_i, introduce a cycle Y_i of $4q$ vertices (variable gadget), add every second vertex on Y_i to X, and label all the other vertices on the cycle alternately with "+" and "−". For each clause c_j, add a star C_j with three leaves (clause gadget) and add its center vertex to X. Each of the three leaves of C_j corresponds to a literal in c_j, and each leaf is connected to a variable gadget as follows. Suppose that l_j^k is a literal y_i or $\neg y_i$, and let a_k be the leaf of C_j corresponding to l_j^k. Add a star with three leaves (connection gadget), identify one leaf with a_i, identify another leaf with an *unused vertex*[5] on Y_i with label "+" if l_j^k is positive and with an unused vertex on Y_i with label "−" if l_j^k is negative, and add the remaining leaf to X. Finally, for each remaining unused vertex v with label "+" or "−" in G, add a star with three leaves (minimality gadget), add two of its leaves to X, and add an edge connecting the center of the star with v. This completes the construction.

Obviously, $G - X$ only contains paths on three vertices as connected components (cf. Figure 6.9), that is, $G - X \in \Pi$. Moreover, X is minimal, that is, for any $v \in X$, $G - (X \setminus \{v\})$ does not satisfy Π. Let r be the number of minimality gadgets. We show that formula F has a satisfying truth assignment if and only if there exists a size-$(r + 3pq + 3q)$ set X', $X' \cap X = \emptyset$, that obstructs all forbidden induced subgraphs in G. In other words, F has a satisfying truth assignment if and only if $(G, X, r + 3pq + 3q)$ is a yes-instance of Size Disjoint Π-Vertex Deletion. The NP-hardness of Disjoint Π-Vertex Deletion then follows

[5]This means that no vertex of another connection gadget has been identified with this vertex on Y_i, that is, it is of degree two.

from Lemma 6.5.

($\Rightarrow$) Assume that a satisfying truth assignment for F is given. Based on this truth assignment, we construct the disjoint solution X', beginning with $X' := \emptyset$, as follows. For each variable y_i, $1 \leq i \leq p$, if y_i = true, then add every vertex on Y_i with label "+" to X', and if y_i = false, then add every vertex on Y_i with label "−" to X'. Add the center vertex of each minimality gadget to X'. For each literal l_j^k of each clause c_j, if l_j^k = true, then add a_k of the corresponding clause gadget C_j to X', and if l_j^k = false, then add the center of the corresponding connection gadget (which is adjacent to a_k) to X'. Clearly, $|X'| = r + 3pq + 3q$. The connected components of $G - X'$ are either isolated vertices or paths on at most three vertices (thus, $G - X' \in \Pi$): the set X' contains the center of each minimality gadget, hence the leaves of the minimality gadgets are isolated vertices in $G - X'$. Concerning a variable gadget Y_i, observe that every fourth vertex on the cycle is in X', and, if a vertex labeled with "+" or "−" is not in X', then its neighbor outside of Y_i (which belongs either to a minimality gadget or to a connection gadget) is in X'. Thus, in $G - X'$, the remaining vertices of Y_i induce connected components that are paths on three vertices. For a connection gadget, observe that either the center vertex is in X' or two of its leaves (which are identified with vertices on other gadgets) are in X', so there remains either an isolated vertex or a single edge in $G - X'$. Concerning a clause gadget C_j, if there is a satisfying truth assignment, then at least one literal is true; therefore, at least one leaf of C_j is in X'. If a leaf is not in X', then its neighbor on the corresponding connection gadget is in X'. Hence, the connected component in $G - X'$ that includes the center of C_j is either a path on three vertices, a single edge, or an isolated vertex (depending on how many literals of c_j are true).

($\Leftarrow$) Let X', $X' \cap X = \emptyset$, be a size-$(r+3pq+3q)$ vertex set that obstructs every forbidden induced subgraph in G. We may assume that X' does not contain any degree-one vertex in G (since a degree-one vertex in X' could simply be replaced by its neighbor). Recall that the set of minimal forbidden induced subgraphs contains the star with three leaves and the path on four vertices. Each minimality gadget is a star with three leaves, and since we assumed that no degree-one vertex is in X', its center vertex must be in X'. Hence, X' contains exactly r vertices of the minimality gadgets. Since P_4s are forbidden, at least every fourth vertex on the cycle of each variable gadget has to be in X'. However, we will see that X' contains exactly three vertices for each clause (thus, $3q$ vertices for all clauses), and these vertices cannot be vertices on any variable gadget. Therefore, for each variable gadget Y_i, the set X' must contain *exactly* every fourth vertex of Y_i (in order to obtain a total number of $3qp$ vertices in X' for all p variable gadgets). Thus X' either contains all vertices labeled "+" or all vertices labeled "−". If X' contains all vertices labeled "+", then we set $y_i :=$ true. If X' contains all vertices labeled "−", then we set $y_i :=$ false. It remains to show that the assignment defined in this way is a satisfying truth assignment for the formula F.

For a clause gadget C_j, and for each leaf a_k of C_j corresponding to literal l_j^k,

let u_k be the center of the corresponding connection gadget, v_k be the degree-one neighbor of u_k, and w_k be the neighbor of u_k on the variable gadget Y_i, for some $1 \leq i \leq p$ (cf. Figure 6.9). There is a P_4 containing the center of C_j, together with a_k, u_k, and v_k. Since the center of C_j is in X, the set X' has to contain at least three vertices to obstruct the three P_4s corresponding to C_j (one for each leaf). Thus, for all clauses, there are at least $3q$ vertices in X' that obstruct these P_4s. In total, X' contains $r+3pq+3q$ vertices. Therefore, there are *exactly* $3q$ vertices in X' that obstruct these P_4s. Thus, for a clause gadget C_j, for each leaf a_k, either $a_k \in X'$ or $u_k \in X'$. Which case applies depends on which vertices from Y_i are in X': if $w_k \notin X'$, then w_k together with u_k and its two neighbors on Y_i induce a star with three leaves, thus $u_k \in X'$. If $w_k \in X'$, then either $a_k \in X'$ or $u_k \in X'$. If $w_k \in X'$ and $u_k \in X'$, however, then one can simply remove u_k from X' and add a_k instead. After that, X' still obstructs all forbidden induced subgraphs. Since X' obstructs all forbidden induced subgraphs, at least one leaf a_k of C_j must be in X', which implies that $w_k \in X'$. Let Y_i be the variable gadget that contains w_k. If w_k has label "+", then y_i = true by the definition of the assignment, and by construction $l_j^k = y_i$ is a positive literal, hence c_j is satisfied. If w_k has label "−", then y_i = false, and, by construction, $l_j^k = \neg y_i$ is a negative literal, hence c_j is satisfied. Summarizing, for every clause there is at least one true literal and thus the constructed truth assignment satisfies F. □

Finally, we consider the case that the path on four vertices is not forbidden.

Lemma 6.10. *If the set $\mathcal{H}$ of minimal forbidden induced subgraphs corresponding to property Π contains a star H with three leaves and $\mathcal{H}$ does not contain the path on four vertices, then* Disjoint Π-Vertex Deletion *is NP-hard.*

Proof. As for Lemma 6.9, the proof is by reduction from 3-CNF-SAT to Size Disjoint Π-Vertex Deletion, and we use the same notation for the 3-CNF-SAT formula as employed there. The proof principle is similar, but the gadgets differ. In the following, we only describe the particularities of this construction and omit straightforward details that can directly be adapted from the proof of Lemma 6.9. An example of the construction is given in Figure 6.10. The basic structure of a variable gadget is a cycle of $4q$ vertices (in the following, further vertices and edges will be added to each such cycle). Add every third vertex on that cycle to X, and for each such vertex $v \in X$ add a new vertex and make it adjacent to v. Then, label the remaining vertices on the cycle, that is, vertices on the cycle that are not in X, alternately with "+" and "−". For each clause c_j introduce a star with three leaves C_j (clause gadget). Each of the three leaves of C_j corresponds to a literal in c_j. As in the proof of Lemma 6.9, we connect the leaves of C_j with vertices labeled with "+" or "−" on the corresponding variable gadgets, depending on whether the corresponding literal is positive or negative, respectively. Herein, for a pair of adjacent labeled vertices, one with label "+" and one with label "−", at most one of them is connected to a clause gadget. The connection gadget is a star with four leaves, one leaf is identified with a leaf

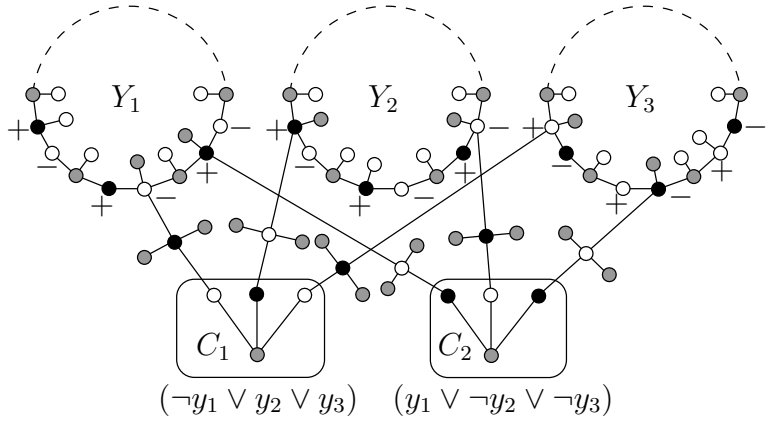


Figure 6.10: Example for the reduction in the proof of Lemma 6.10 for the 3-CNF-SAT formula $(\neg y_1 \vee y_2 \vee y_3) \wedge (y_1 \vee \neg y_2 \vee \neg y_3)$. The vertices in the given solution X are gray, the vertices in the disjoint solution X' corresponding to the satisfying truth assignment $y_1 = \text{true}$, $y_2 = \text{true}$, $y_3 = \text{false}$, are black.

of a clause gadget, one leaf is identified with a vertex labeled "+" or "−" on a variable gadget, and the remaining two leaves are added to X. After adding all connection gadgets, for each pair u, v of adjacent labeled vertices, one with label "+" (say, u) and one with label "−" (say, v), if both u and v have not been connected to a clause gadget, add a new vertex and make it adjacent to u. Moreover, for each labeled vertex u on a variable gadget that has been connected to a clause gadget, add a new vertex, add it to X, and make it adjacent to u.

For the resulting graph G, observe that $G - X$ contains only isolated vertices, paths on three vertices, and paths on four vertices as connected components. Only the paths on four vertices have a higher α-sequence than the star with three leaves, but, by the preconditions of Lemma 6.10, these are not forbidden. Therefore, X obstructs all forbidden induced subgraphs in G, and thus $G - X \in \Pi$. Moreover, X is minimal, since for each $v \in X$, $G - (X \setminus \{v\})$ contains a star with three leaves. We claim that F has a satisfying assignment if and only if there exists a size-$(3q + 3pq)$ set X', $X' \cap X = \emptyset$, that obstructs all forbidden induced subgraphs in G. The proof of the claim is very similar to the proof of Lemma 6.9. For this reason, we omit the details. Note that for each variable gadget (together with the degree-one vertices that have been added) the only possibilities are that all vertices labeled "+" or all vertices labeled "−" can be in X', and for each clause, X' must contain exactly three vertices, one for each literal. For each clause gadget in G, at least one leaf must be in X', and thus the clause gadgets are obstructed. This guarantees the satisfiability of the corresponding formula F if X' has size $3q + 3pq$. □

Clearly, Lemmas 6.8–6.10 yield Theorem 6.4.

6.4 Outlook

We completely settled the complexity of the Disjoint Compression Task in the case of vertex deletion problems on undirected graphs for a non-trivial hereditary property determined by the components. There are important problems amenable to iterative compression that do not fall into the problem class studied here. Among these, in particular, we have Directed Feedback Vertex Set and Almost 2-Sat. Hence, it would be interesting to further generalize our results to other problem classes, among these also being vertex deletion problems on directed graphs or bipartite graphs and edge deletion problems. Our work here has left open the case where a forbidden subgraph may consist of more than one connected component. Finally, we also did not explore the case when one demands that the given solution X in Disjoint Π-Vertex Deletion is already optimal (and not just inclusion-minimal), or close to optimal (for instance, that the given solution contains only one more vertex compared to an optimal solution).

The trick of enumerating all two-partitions of the given solution into one part to keep and one part to exchange, which we employed for the general iterative compression framework for Π-Vertex Deletion in Section 6.2, has been applied in almost all known applications of iterative compression. Thus, if Disjoint Π-Vertex Deletion is NP-hard, then it seems difficult to reach the "$O(2^k)$-barrier" for an iterative compression based algorithm for the corresponding Π-Vertex Deletion problem. To overcome this problem, a more elaborated enumeration of only "relevant" two-partitions would be necessary. A first example of such an approach has been given for Vertex Cover, yielding an iterative compression based algorithm with running time $O(1.443^k \cdot mn\sqrt{n})$ [Pei07]. This approach relies also on the fact that Vertex Cover can be solved in polynomial time in bipartite graphs (see Chapter 1). However, unfortunately, all other vertex deletion problems for hereditary graph properties that are determined by the components, restricted to bipartite graphs, are NP-complete [Yan81a], and as a consequence it seems rather difficult to generalize this approach.

An exception of the general trick to enumerate all two-partitions of the given solution is a recent new version of the proof that Vertex Bipartization is fixed-parameter tractable, where the trick is to enumerate *three-partitions* of the given solution [LSS09]. Like for the other proofs [RSV04, Hüf09], the exponential part of the running time is 3^k, because with a three-partition of the given solution, the remaining task can be solved in polynomial time with maximum flow techniques. For the moment, this is the only example of an iterative compression approach with "three-partitions", but if this turns out to work for other problems as well, then an analogous study of the complexity of the corresponding compression task would be interesting.

Chapter 7

Graph Packing

In this chapter, we consider the problem of packing at least k vertex-disjoint copies of a fixed graph H into a given graph. This problem, called H-Packing, is a generalization of Maximum Matching, which can be regarded as the task of packing a maximum number of vertex-disjoint edges. The graph packing problem is fixed-parameter tractable with respect to the maximum number of copies as the parameter. We show a problem kernel of $O(k^{|V(H)|-1})$ vertices. Based on the kernelization technique, we also give a new version of a problem kernel for Hitting Set.

The problem kernel for H-Packing is the main result of this chapter and is presented in Section 7.2. Then, we show in Section 7.3 how the kernelization technique can be applied for Hitting Set.

7.1 Introduction and Known Results

The H-Packing problem, sometimes also called H-Matching, is defined as follows.

H-Packing
Input: An undirected graph $G = (V, E)$ and an integer $k \geq 0$.
Question: Does G contain k vertex-disjoint copies of H?

A solution to H-Packing is a packing of k vertex-disjoint copies of H; we call such a solution an *H-packing*.

Applications. Packing problems have various applications in computational biology [ABWB+09]; for instance, the so-called Full Sibling Reconstruction problem is a packing problem [ABWB+09]. Packing problems have many applications ranging from information theory to the design of efficient statistical experiments [Yus07]. Triangle Packing, that is, K_3-Packing, has applications in genome rearrangement problems [ABWB+09, CR02].

Polynomial-Time Solvable Cases. If H is a single edge, then the H-Packing problem is equivalent to Maximum Matching, and therefore polynomial-time solvable. H-Packing is linear-time solvable on graphs of bounded treewidth; this can be shown based on a monadic second-order logic (MSO) formulation of H-Packing [Yus07] (see Section 8.2.3 for an example of an MSO formulation). Furthermore, Triangle Packing is polynomial-time solvable on split graphs, cographs [GRC+98], and on graphs of maximum degree three [CR02].

Hardness and Approximation. If H is a connected graph with at least three vertices, then H-Packing becomes NP-complete [KH78] and also APX-complete [Kan94]. There exists a simple polynomial-time approximation algorithm that greedily finds an inclusion-maximal set of copies of H in G, yielding an approximation factor of $|V(H)|$ (cf. [Yus07]). For every fixed $t \geq 3$, K_t-Packing can be approximated in polynomial time within a factor of $t/2 + \epsilon$, for any $\epsilon > 0$ [HS89]. H-Packing remains NP-complete on planar graphs [BJL+90], but there exists a polynomial-time approximation scheme (PTAS) [Bak94].

There exist numerous results if H is a triangle. Triangle Packing is NP-complete on chordal, line, total graphs [GRC+98], and planar graphs of maximum degree four [CR02]. Concerning approximation, in general Triangle Packing is APX-hard [CR02]. There exists a factor-1.2 polynomial-time approximation on graphs of maximum degree four [MW08]. On general graphs, it can be approximated in polynomial time within a factor of $3/2 + \epsilon$ for any $\epsilon > 0$ [HS89]. However, it cannot be approximated in polynomial time with a ratio better than 95/94 [CC06] (assuming P $\neq$ NP), or with a ratio better than 76/75 (assuming RP $\neq$ NP) [ABWB+09].

The above is only a very brief description of the main results; for a more detailed exposition concerning H-Packing we refer to the survey by Yuster [Yus07].

There are numerous related problems, for instance, packing of graphs from a given family $\mathcal{H}$ (each copy must be isomorphic to some $H \in \mathcal{H}$, packing on directed graphs, factor problems (every vertex must be part of some copy of H), edge-disjoint packing, and many more (see [Yus07]).

Parameterized Complexity. H-Packing can be solved in $2^{|V(H)|k} \cdot \text{poly}(n)$ time with a randomized algorithm [Kou08] and in $2^{2|V(H)|k} \cdot \text{poly}(n)$ time [CKL+09] with a deterministic algorithm, which has later been improved to $2^{(2|V(H)|-1)k} \cdot \text{poly}(n)$ [FLLW09]. The deterministic algorithms also work for a weighted variant of the problem within the same running time bounds. Hence, H-Packing is fixed-parameter tractable with respect to the parameter k.

If H is a triangle, then there exists a problem kernel of $108k^3 - 73k^2 - 18k$ vertices [FHR+04]. If H is a path on three vertices, then there exists a problem kernel with $15k$ vertices [PS06], which has been improved to $7k$ vertices [WNFC08]; the best-known parameterized algorithm runs in $2.448^{3k} \cdot \text{poly}(n)$ time [FR09]. If H is a star with a constant number of leaves, then H-Packing admits an $O(k^2)$-vertex

problem kernel [PS06]. On a connected cubic graph with $n > 16$ a P_3-packing consists of at least $117/152\ n$ vertices [KMZ08]. This directly gives a linear-vertex problem kernel for P_3-Packing on connected cubic graphs.

An interesting question in parameterized complexity is the existence of lower bounds for kernel sizes. Up to now, we are only aware of lower bounds on the constant factor of a linear kernel (that is, of size ck for some constant c) for Vertex Cover, Independent Set, and Dominating Set on planar graphs [CFKX07]. Moreover, it is known that several problems do not admit a polynomial-size kernel [BDFH09, FS08, DLS09] (the existence of polynomial-size kernels for these problems would imply the collapse of the polynomial hierarchy to the third level). However, to the best of our knowledge, there is no lower bound example of a problem that does admit a polynomial-size kernel, but no linear-size kernel. Fellows et al. [FHR+04] conjectured that K_r-Packing, that is, packing cliques on r vertices, might be a candidate for a problem whose kernel cannot be smaller than $O(k^r)$ vertices. However, our result for H-Packing directly shows that an $O(k^{r-1})$-vertex kernel is possible. Our technique differs from the technique used by Fellows et al. [FHR+04] in how we analyze an initially computed greedy packing of triangles based on which the size bound of the whole kernel is derived. The drawback of their method is, as they state, that it is not obvious how to generalize it to H-Packing. Our approach combines ideas from an improved kernelization of Hitting Set [Abu09] and from problem kernels for generalized matching and set packing problems [FKN+07] to achieve this. The $O(k^3)$-vertex kernel by Fellows et al. [FHR+04] is based on *crown decompositions*; while we also apply the idea behind crown decompositions, we will see that it is actually not necessary to compute the decomposition to derive the kernel. This does not improve on the worst-case running time of the kernelization, but might be interesting for practical purposes and would probably also work similarly for other applications of the crown decomposition technique.

7.2 Problem Kernelization for H-Packing

In this section, we improve a bound of $108k^3 - 73k^2 - 18k$ vertices for a problem kernel for Triangle Packing [FHR+04] to a bound of $45k^2$ vertices (Section 7.2.1). Moreover, our approach can be generalized to a problem kernel of $O(k^{|V(H)|-1})$ vertices for H-Packing, where H is a fixed connected graph (Section 7.2.2).

7.2.1 Quadratic-Vertex Problem Kernel for Triangle Packing

In this section, we give a problem kernel with $45k^2$ vertices for Triangle Packing. The problem kernel with $O(k^3)$ vertices by Fellows et al. [FHR+04] starts

with a greedy packing $\mathcal{P}$ of triangles, which contains less than $3k$ vertices (otherwise, we already have a packing of k triangles). Then, based on the size of $\mathcal{P}$, the number of vertices in $V \setminus V(\mathcal{P})$ is bounded, which implies that the total number of vertices in the graph is bounded, yielding a problem kernel. In this sense, $\mathcal{P}$ is a witness for the number of vertices in the graph. The limitation of this approach is that there is too much structure of the graph "outside" of $\mathcal{P}$; we only know that $G - V(\mathcal{P})$ is a triangle-free graph. To deal with this problem, we use a different notion of witness, which contains more triangles than $\mathcal{P}$, but which is still small enough in order to obtain a better bound on the number of vertices in the problem kernel. In our case, the vertices outside of the witness form an independent set, whose size is much easier to bound. Our kernelization is based on the same reduction rules as the kernel by Fellows et al. [FHR+04]. However, our approach applies them differently, and, most importantly, it uses a different analysis. One of the main advantages of our approach is that it is easier to generalize to H-Packing for arbitrary connected graphs H.

Our approach works with the set of all triangles in G. The set of all triangles in a graph can be computed in $O(m\sqrt{m})$ time [AYZ97]. First, we apply the following simple data reduction rule, which is obviously correct and can be exhaustively applied in $O(n + m)$ time.

Reduction Rule 7.1. *Remove all vertices and edges that are not contained in any triangle in G.*

In the following, assume that G is reduced with respect to Reduction Rule 7.1. The general strategy of our kernelization algorithm is as follows. First, we compute in polynomial time a set of not necessarily disjoint triangles $\mathcal{T}$, and we show that if there are sufficiently many vertices in $V(\mathcal{T})$, then the input instance is a yes-instance, and a corresponding size-k packing can be computed in polynomial time. If not, then, with the size bound on $V(\mathcal{T})$, one can bound the size of $V \setminus V(\mathcal{T})$ by applying a data reduction rule based on matching techniques. In this sense, the set $\mathcal{T}$ is the basis of our kernelization and is the *witness* for the size of the kernel.

The witness $\mathcal{T}$ is defined as follows.

Definition 7.1. *A* witness $\mathcal{T}$ *is a maximal set of triangles in G that pairwisely intersect in at most one vertex.*

We will later show that $I := V \setminus V(\mathcal{T})$ forms an independent set.

Lemma 7.1. *A witness $\mathcal{T}$ can be computed in $O(m\sqrt{m})$ time.*

Proof. Compute the set $\mathcal{T}^*$ of all triangles in G in $O(m\sqrt{m})$ time [AYZ97]. The witness can be computed by an iterative algorithm that starts with an empty set $\mathcal{T}$, and then repeats the following until $\mathcal{T}^* = \emptyset$.

Select an arbitrary triangle $T \in \mathcal{T}^*$ and test whether T intersects with each triangle in $\mathcal{T}$ in at most one vertex. If so, then add T to $\mathcal{T}$ and remove it from $\mathcal{T}^*$; otherwise, just remove T from $\mathcal{T}^*$.

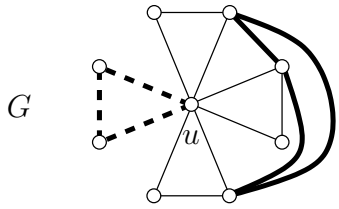


Figure 7.1: In this example, we assume that $k = 2$ and illustrate that a size-$(k-1)$ packing of triangles in $G - u$ implies a size-k packing of triangles in G. A packing of $k - 1 = 1$ triangle in $G - u$ (bold edges) contains three vertices; hence it can contain vertices of at most three other triangles in G that contain u. Therefore, at least one triangle in G that contains u is left (dashed edges), which can be added to the packing, obtaining a packing of two triangles for G.

To be able to efficiently test whether T intersects with each triangle in $\mathcal{T}$ in at most one vertex, maintain a copy of G in which all edges that are a part of a triangle in $\mathcal{T}$ are marked. Thus, T intersects with each triangle in $\mathcal{T}$ in at most one vertex if and only if $E(T)$ contains no marked edge. This can be tested in constant time for each $T \in \mathcal{T}^*$ (assuming that an edge is stored in the graph data structure with a flag that indicates whether it is marked or not and that a triangle is represented by its edges; a simple adjacency list data structure can be easily adapted accordingly).

Altogether, we have $O(m\sqrt{m})$ triangles, and the algorithm described above needs only constant time per triangle. □

In the following, ComputeWitness denotes an algorithm computing a witness. After computing the witness $\mathcal{T}$, the following data reduction rule due to Fellows et al. [FHR+04] is applied.

Reduction Rule 7.2 ([FHR+04])**.** *If there is a vertex $u \in V(\mathcal{T})$ such that there exist at least $3k-2$ triangles in $\mathcal{T}$ that pairwise intersect exactly in u, then delete u from G and set $k := k - 1$.*

To see the correctness of this rule, one has to show that (G, k) is a yes-instance if and only if $(G - u, k - 1)$ is a yes-instance. Let P be a packing of k triangles in G. As a consequence, since u can be in at most one triangle in P, there exists a packing of at least $k-1$ triangles in $G-u$. To see the other direction, let P be a packing of $k-1$ triangles in $G-u$. The packing P contains $3k-3$ vertices, that is, $|V(P)| = 3k-3$. Hence, in G the packing P can overlap with at most $3k-3$ triangles of the at least $3k-2$ triangles that pairwise intersect exactly in u. As a consequence, there is at least one triangle left that can be added to P, obtaining a packing of k triangles in G (see Figure 7.1 for an example). Hence, Reduction Rule 7.2 is correct.

Note that Fellows et al. [FHR+04] start with a vertex-disjoint packing of triangles as "witness" instead of a packing of triangles that pairwise overlap in

at most one vertex, which we use as the witness $\mathcal{T}$. Fellows et al. [FHR+04] then apply Reduction Rule 7.2 on triangles that contain exactly one vertex of their "witness". They find these triangles by computing a maximal matching in the graph resulting by deleting the "witness". In contrast, we use Reduction Rule 7.2 only for reducing vertices that are part of the witness; the vertices "outside" of the witness need not to be considered by Reduction Rule 7.2 with our approach. We have to show that Reduction Rule 7.2 can be efficiently implemented using our approach as well.

Lemma 7.2. *One application of Reduction Rule 7.2 can be performed in $O(m\sqrt{m})$ time.*

Proof. There are at most $O(m\sqrt{m})$ triangles in the witness $\mathcal{T}$. By simply maintaining a counter for each vertex in $V(\mathcal{T})$, one can count the number of triangles each vertex is part of by iterating over all triangles in $\mathcal{T}$ in $O(m\sqrt{m})$ time. If a counter of a vertex reaches at least $3k-2$, then the corresponding vertex can be deleted as described in Reduction Rule 7.2. □

If Reduction Rule 7.2 applies, then the kernelization algorithm restarts with exhaustively applying Reduction Rule 7.1 and calling ComputeWitness. This is repeated until Reduction Rule 7.2 does not apply or until $k = 0$. In total, Reduction Rule 7.2 is called at most k times, and each time Reduction Rule 7.1 and ComputeWitness are called. Therefore, the time for the exhaustive application of Reduction Rule 7.2 is $O(k(m\sqrt{m}+n))$. If $k = 0$ after the exhaustive application of the reduction rules, then G is a yes-instance, thus the kernelization algorithm returns "yes". In the following, we can therefore assume that Reduction Rule 7.1 and Reduction Rule 7.2 do not apply and that $k > 0$, that is, that (G, k) is reduced with respect to these two reduction rules. The next lemma states some important properties of a reduced instance.

Lemma 7.3. *Let (G, k) be an instance of* Triangle Packing *with corresponding witness $\mathcal{T}$ such that Reduction Rule 7.1 and Reduction Rule 7.2 do not apply. Then, the following holds:*

1. *The set $I := V \setminus V(\mathcal{T})$ forms an independent set in G and*
2. *each triangle that contains a vertex from I shares an edge with a triangle from $\mathcal{T}$.*

Proof. (1.) Suppose that I is not an independent set in G. Let e be an edge in $G[I]$. Due to Reduction Rule 7.1, the edge e must be contained in a triangle T in G such that $T \notin \mathcal{T}$. The triangle T intersects each triangle in $\mathcal{T}$ in at most one vertex, thus T is added to $\mathcal{T}$ by ComputeWitness, contradicting $T \notin \mathcal{T}$. (2.) If a triangle $T \notin \mathcal{T}$ contains a vertex from I but shares no edge with a triangle from $\mathcal{T}$, then again T intersects each triangle from $\mathcal{T}$ in at most one vertex, contradicting $T \notin \mathcal{T}$. □

The structure of a reduced graph G is illustrated in Figure 7.3. Note that the graph $G[V(\mathcal{T})]$ might contain edges that are not part of any triangle in $\mathcal{T}$; however, by Lemma 7.3, these edges are not part of any triangle that contains a vertex from I. Hence, there are only $O(|\mathcal{T}|)$ edges in $G[V(\mathcal{T})]$ that are part of triangles that contain vertices from I. This fact is crucial to obtain a quadratic bound on the number of vertices for our problem kernel. To this end, we need to bound the number of vertices in $V(\mathcal{T})$ and the number of triangles in $\mathcal{T}$.

Lemma 7.4. *Let (G, k) be an instance of* Triangle Packing *with corresponding witness $\mathcal{T}$ such that Reduction Rule 7.1 and Reduction Rule 7.2 do not apply. If $|V(\mathcal{T})| > 18k^2$ or if $|\mathcal{T}| > 9k^2$, then G contains k vertex-disjoint triangles.*

Proof. Assume that there do not exist k vertex-disjoint triangles in $\mathcal{T}$. Let $\mathcal{P} \subseteq \mathcal{T}$ be a maximum-size set of vertex-disjoint triangles. Thus, $|\mathcal{P}| \leq k - 1$, and for each triangle $T \in \mathcal{T} \setminus \mathcal{P}$ we know that $V(T) \cap V(\mathcal{P}) \neq \emptyset$. By the definition of witness (Definition 7.1), the triangles in $\mathcal{T}$ pairwise intersect in a most one vertex. For each triangle $T \in \mathcal{P}$, due to Reduction Rule 7.2, each vertex in T is contained in at most $3k - 3$ triangles, thus for each vertex $v \in V(T)$ we have at most $6k-6+1 \leq 6k$ vertices contained in triangles that contain v. Thus, in total we have at most $3|\mathcal{P}| \cdot 6k \leq 18k^2$ vertices and at most $3|\mathcal{P}| \cdot 3k \leq 9k^2$ triangles in $\mathcal{T}$. Therefore, if $|V(\mathcal{T})| > 18k^2$ or $|\mathcal{T}| > 9k^2$, then G contains k vertex-disjoint triangles. □

If $|V(\mathcal{T})| > 18k^2$ or $|\mathcal{T}| > 9k^2$, then these k vertex-disjoint triangles can be found by a greedy algorithm that selects an arbitrary triangle, deletes all other intersecting triangles, and proceeds recursively with the remaining instance until it has found k triangles.

Thus, our kernelization algorithm outputs "yes-instance" if one of the conditions of Lemma 7.4 applies. If this is not the case, then it remains to upper-bound the size of I. The basic observations to achieve this are that the maximum number of triangles in a triangle packing that contain vertices from I depends on the number of edges in $E(\mathcal{T})$ (by Lemma 7.3 each triangle that contains a vertex from I has to contain an edge from $E(\mathcal{T})$), and that some vertices from I are superfluous; see Figure 7.2 for an example. In this example, the vertex labeled "a" is superfluous, because one can replace any triangle packing by another triangle packing of the same size that does not contain "a". This replacement works because one can pair each of the vertices "b" and "c" with a particular edge in $E(\mathcal{T})$, respectively. In other words, we can find a matching between vertices in I and edges in $E(\mathcal{T})$, and all unmatched vertices are superfluous and can be removed from the graph. Since the number of edges in $\mathcal{T}$ is bounded, the number of non-superfluous vertices in I is bounded as well. In the following, we show that this approach is correct.

To this end, we define an auxiliary bipartite graph $G_{\mathcal{T}}$ as follows. The vertex set consists of I as one partite set and $J := \{v_e \mid e \in E(\mathcal{T})\}$ as the other, and $G_{\mathcal{T}}$ contains an edge $\{u, v_e\}$ if $\{u\} \cup e$ induces a triangle in G. Note that by part (2)

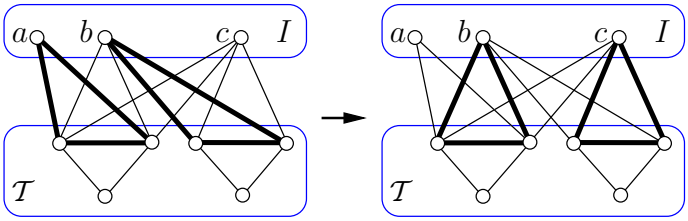


Figure 7.2: Left: Graph with witness $\mathcal{T}$ and a packing of two triangles (bold edges). Right: Packing of two triangles that does not contain the vertex labeled "a" in I.

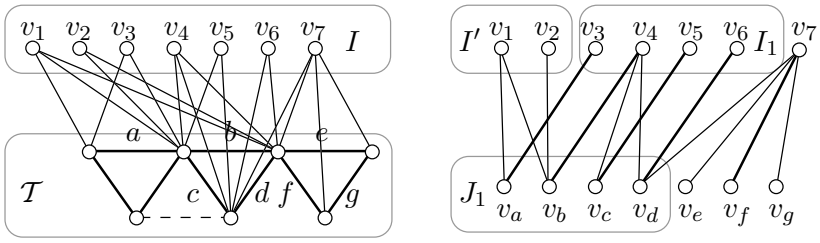


Figure 7.3: Left: Graph with witness $\mathcal{T}$. The edge set $E(\mathcal{T})$ of the witness is drawn bold. The dashed edge is not contained in any triangle that contains a vertex from I nor in any triangle in the witness. Right: Corresponding auxiliary graph. Note that degree-0 vertices (corresponding to unlabeled edges of $G[V(\mathcal{T})]$) are not drawn, because they are not part of any triangle that contains a vertex from I. The bold edges are in a maximum matching. For the definition of the vertex sets see the proof of Lemma 7.5.

of Lemma 7.3 every triangle containing a vertex in I is "represented" by an edge in $G_{\mathcal{T}}$. See Figure 7.3 for an example. Since $\mathcal{T}$ contains $O(m\sqrt{m})$ triangles, $G_{\mathcal{T}}$ can be constructed in $O(m\sqrt{m})$ time. With the help of this auxiliary graph, we can state a data reduction rule to upper-bound the size of I.

Reduction Rule 7.3. *Compute a maximum matching in $G_{\mathcal{T}}$. Remove all unmatched vertices in I from G.*

Lemma 7.5. *Reduction Rule 7.3 is correct, that is, G has a size-k packing of triangles if and only if the graph resulting by removing all unmatched vertices in I from G has a size-k packing of triangles.*

Proof. Let M be the computed maximum matching in $G_{\mathcal{T}}$ and let I' be all unmatched vertices in I (see Figure 7.3). Since M is maximum, the graph $G_{\mathcal{T}}$ contains no M-augmenting path. We have to show that G contains k vertex-disjoint triangles if and only if $G - I'$ contains k vertex-disjoint triangles.

($\Leftarrow$) This direction is trivial, since a set of k vertex-disjoint triangles in $G - I'$ is also contained in G.

($\Rightarrow$) Let $\mathcal{P}$ be a set of k vertex-disjoint triangles in G. If no triangle in $\mathcal{P}$ contains a vertex of I', then $\mathcal{P}$ is a set of k vertex-disjoint triangles in $G - I'$. Therefore, suppose that there is a triangle in $\mathcal{P}$ that contains a vertex of I'. We show in the following that we can always modify $\mathcal{P}$ such that there is no triangle containing a vertex of I'.

Let $I_1 \subseteq I \setminus I'$ be the set of vertices in $I \setminus I'$ to which there exists an M-alternating path from some vertex in I' (see Figure 7.3). Each vertex $u \in I_1$ is an endpoint of an edge in M because there is an M-alternating path from some vertex $w \in I'$ to u, and the path begins with an edge that is not contained in M (since all vertices in I' are unmatched). Let $M' \subseteq M$ be the matching edges that have an endpoint in I_1, and let $J_1 := J \cap V(M')$ be the corresponding second endpoints of edges from M' (see Figure 7.3). We claim that every triangle that contains a vertex from $I' \cup I_1$ contains an edge e corresponding to a vertex $v_e \in J_1$. As we will see, this implies that there exists a triangle packing that does not contain any vertex from I'.

To show the claim, let T be a triangle in $\mathcal{P}$ that contains a vertex $u \in I'$. Suppose that T contains an edge e corresponding to a vertex v_e in $J \setminus J_1$. Since $v_e \notin J_1$, we know that v_e is not matched by M; otherwise, there would be an M-alternating path (u, v_e, w) for some vertex $w \in I \setminus I'$, and this would imply $v_e \in J_1$ by the definition of J_1. Therefore, $\{u, v_e\}$ could be added to M, contradicting that M is maximum. Similarly, every triangle T in $\mathcal{P}$ that contains a vertex $u \in I_1$ contains an edge e corresponding to some $v_e \in J_1$. To see this, assume again that $v_e \in J \setminus J_1$. Then, v_e must be unmatched, but then the path in $G_{\mathcal{T}}$ consisting of the M-alternating path from some vertex $w \in I'$ to u and the edge $\{u, v_e\}$ forms an M-augmenting path, contradicting that M is maximum. This shows the claim.

Since M' is a perfect matching between I_1 and J_1 (that is, every vertex in $I_1 \cup J_1$ is matched and every matching edge has one endpoint from I_1 and the other from J_1), we can always replace all triangles in $\mathcal{P}$ that contain vertices in $I' \cup I_1$ by the same number of triangles containing only vertices in I_1. This shows that $G[V \setminus I']$ also contains k vertex-disjoint triangles. □

Concerning the running time of Reduction Rule 7.3, observe that $G_{\mathcal{T}}$ contains $O(m)$ vertices and $O(m\sqrt{m})$ edges. The computation of a maximum matching in this graph can therefore be performed in $O(m^2)$ time using the algorithm by Hopcroft and Karp [HK73] (running time $m\sqrt{n}$ for an n-vertex and m-edge bipartite graph).

Lemma 7.6. *After applying Reduction Rule 7.3, at most $27k^2$ vertices of I remain.*

Proof. By Lemma 7.4, the witness $\mathcal{T}$ computed by ComputeWitness contains at most $9k^2$ triangles. Since $J := \{v_e \mid e \in E(\mathcal{T})\}$, we know that $|J| \leq 27k^2$. Due to Reduction Rule 7.3, all remaining vertices of I are matched by a maximum matching between I and J in $G_{\mathcal{T}}$. Therefore, there remain at most $27k^2$ vertices of I. □

Theorem 7.1. Triangle Packing *admits a problem kernel of at most* $45k^2$ *vertices, which can be constructed in* $O(k(m\sqrt{m}+n)+m^2)$ *time.*

Proof. By Lemma 7.4, we have at most $18k^2$ vertices in $V(\mathcal{T})$ and, by Lemma 7.6, there remain at most $27k^2$ vertices of I, thus in total we have at most $45k^2$ vertices. The running time is simply the total time for exhaustively applying Reduction Rule 7.1 and Reduction Rule 7.2, constructing a witness structure, constructing the auxiliary graph, computing a maximum matching in it, and deleting the unmatched vertices. □

7.2.2 Generalizing to H-Packing

In this section, we generalize the kernelization approach for Triangle Packing to H-Packing for an arbitrary connected graph H. The main difference to Triangle Packing is a new reduction rule that bounds the size of the witness and generalizes Reduction Rule 7.2. Let h denote the number of vertices in H. Note that h is a constant. We start with a trivial reduction rule analogous to Reduction Rule 7.1.

Reduction Rule 7.4. *Remove all vertices and edges that are not contained in any copy of* H *in* G.

Lemma 7.7. *Reduction Rule 7.4 is correct and can be performed in* $O(n^h \cdot h^2)$ *time.*

Proof. The correctness is trivial. An algorithm with the claimed running time works as follows. Iterate over the at most $\binom{n}{h}$ many copies of H in the input graph and mark all vertices and edges in the graph G that are contained in some copy in $O(h^2)$ time. After that, the unmarked vertices and edges in G are not contained in any copy of H and can be removed. The total running time is $O(n^h \cdot h^2)$. □

In the following, we assume that G is reduced with respect to Reduction Rule 7.4. Let $\mathcal{H}$ be the set of all copies of H in G. Due to Reduction Rule 7.4, every vertex in G is contained in at least one copy of H in $\mathcal{H}$. The set $\mathcal{H}$ can be computed in $O(n^h \cdot h^2)$ time by simply trying all $\binom{n}{h}$ vertex subsets and testing in $O(h^2)$ time whether the chosen subset is a copy of H. The kernelization algorithm follows the ideas of the problem kernel for Triangle Packing in Section 7.2.1. As for Triangle Packing, we define a witness; here, the witness is a set of copies of H that pairwise overlap in at most $h-2$ vertices. As we will see, as for Triangle Packing, the vertices that are not part of the witness form an independent set. See Figure 7.4 for an illustration. The basic outline of the kernelization algorithm is the same as for Triangle Packing, that is, that we first have to bound the size of the witness by some appropriate reduction rule, and then, based on that size bound, to bound the number of vertices in I by removing superfluous vertices from I. The most involved part of the kernelization algorithm

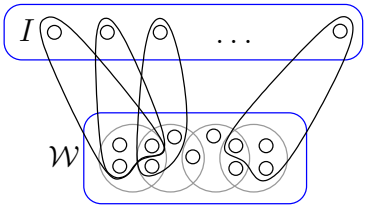


Figure 7.4: Example for $h = 4$ of the general structure of a graph with a witness $\mathcal{W}$ (gray cycles). The witness is a maximal packing of copies of H that pairwise overlap in at most two vertices. The remaining copies of H, which contain a vertex from $I := V \setminus V(\mathcal{W})$, overlap with some copy of H in $\mathcal{W}$ in three vertices.

is the reduction rule that will bound the size of the witness. For an intuitive idea of the following witness definition and the description of the kernelization algorithm, assume that there are copies of H that pairwise intersect in the vertex set T. Let $\mathcal{C}$ be the set of these copies of H. For Triangle Packing, we had $|T| = 1$, and due to the definition of the witness, we knew that the triangles pairwise intersect *exactly* in T, and we could safely assume that one of these triangles is always part of an optimal triangle packing if $\mathcal{C}$ is big enough and therefore delete T from G (Reduction Rule 7.2). However, for general H, this idea does not work anymore, since in general $|T| > 1$, and then we cannot simply decide to delete T, because we cannot determine which vertex of T is part of a maximum-cardinality H-packing. Still, if there are too many graphs (the exact bound depends on the size of T) in $\mathcal{C}$, then we can find a subset $\mathcal{C}' \subsetneq \mathcal{C}$ such that there always exists an optimal H-packing that does not include any copy of H from $\mathcal{C}'$. These copies do not have to be considered for finding an optimal solution. Therefore, all vertices that are only contained in such "unnecessary" copies of H can be removed from the graph. A witness does not contain these "unnecessary" copies of H; as a consequence, a witness is defined with respect to a subset $\mathcal{H}'$ of $\mathcal{H}$.

Definition 7.2. *Let $\mathcal{H}$ be the set of all copies of H in G. A* witness *with respect to a set $\mathcal{H}' \subseteq \mathcal{H}$ for H-Packing is a maximal subset $\mathcal{W} \subseteq \mathcal{H}'$ such that the copies of H in $\mathcal{W}$ pairwise intersect in at most $h - 2$ vertices.*

An example of a witness is given in Figure 7.4 for the case $h = 4$.

The algorithm ReducedWitness, given in Figure 7.5, computes a witness $\mathcal{W}$ with respect to $\mathcal{H} \setminus \mathcal{R}$, where $\mathcal{R}$ is a set of "unnecessary" copies of H in $\mathcal{H}$, that is, if there exists a size-k H-packing, then there is a size-k H-packing that does not use any element of $\mathcal{R}$. The identification of unnecessary copies of H is derived from a combination of ideas for data reduction rules for Hitting Set [Abu09] and generalized matching and set cover problems [FKN+07]. The algorithm uses an iterative approach, starting with empty sets $\mathcal{R}$ and $\mathcal{W}$. In line 3 of Figure 7.5, it computes a witness $\mathcal{W}$ with respect to $\mathcal{H} \setminus \mathcal{R}$. In lines 5–11, the algorithm

Algorithm: ReducedWitness ($\mathcal{H}$)
Input: A set $\mathcal{H}$ of copies of H in G.
Output: A set $\mathcal{R}$ of unnecessary copies of H and a witness $\mathcal{W}$ with respect to $\mathcal{H} \setminus \mathcal{R}$.

```
1  R ← ∅; W ← ∅
2  repeat
3      Greedily add elements from H \ (W ∪ R) to W such that W is a witness.
4      C' ← ∅
5      for i ← 0 to h − 3 do
6          for each H' ∈ W do
7              for each T ⊊ V(H'), |T| = h − 2 − i do
8                  C ← {H'' ∈ W | V(H'') ⊇ T}
9                  if |C| > Σ_{t=0}^{i+1} (hk)^t then
10                     choose any set C' ⊊ C of size |C| − Σ_{t=0}^{i+1} (hk)^t.
11                     W ← W \ C'; R ← R ∪ C'
12 until C' = ∅
13 return W, R
```

Figure 7.5: Pseudo-code of the algorithm to compute the witness $\mathcal{W}$.

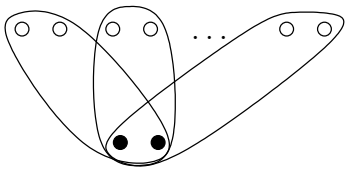
identifies unnecessary copies and adds them to $\mathcal{R}$; the correctness of this part will be shown with Lemma 7.8. After having identified and removed unnecessary copies of H from $\mathcal{W}$, the set $\mathcal{W}$ might not be a witness with respect to $\mathcal{H} \setminus \mathcal{R}$; therefore, the algorithm repeats until no more unnecessary copies of H can be found. Then, the resulting set $\mathcal{W}$ is a witness with respect to $\mathcal{H} \setminus \mathcal{R}$, since it is updated in line 3 before ReducedWitness returns.

Let $\mathcal{W}$ be the witness that is returned by ReducedWitness($\mathcal{H}$). The following lemma shows that there exists a maximum H-packing in G that does not contain any copy of H that is removed from $\mathcal{W}$ and added to $\mathcal{R}$ in line 11 of ReducedWitness. After executing ReducedWitness, the set $\mathcal{R}$ contains all removed copies of H.

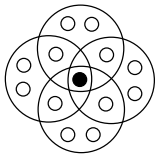
Lemma 7.8. *If there exists an H-packing $\mathcal{P}$ of size k in G, then there exists an H-packing $\mathcal{P}'$ of size k in G that does not contain any element of $\mathcal{R}$.*

Proof. If $\mathcal{R} \cap \mathcal{P} = \emptyset$, then $\mathcal{P}' := \mathcal{P}$ is an H-packing of size k that does not contain any copy of H from $\mathcal{R}$. Hence, assume that $\mathcal{R} \cap \mathcal{P} \neq \emptyset$. We show that we can replace each copy of H in $\mathcal{R} \cap \mathcal{P}$ by another copy of H not contained in $\mathcal{R}$ such that the resulting packing has size k, which proves the lemma.

We show the claim by induction on i (line 5 in Figure 7.5). Intuitively, i determines the size of the vertex set T computed in line 7. For $i = 0$, T contains $h - 2$ vertices, thus all copies of H whose vertex sets are supersets of T intersect exactly in T, because the copies of H pairwisely overlap in at most $h-2$ vertices (Definition 7.2). As shown below, the number of these copies can be



(a) Case $|T| = h-2 = 2$. The copies of H in $\mathcal{T}$ that contain T pairwisely intersect exactly in T.



(b) Case $|T| = h - 3 = 1$. The copies of H in $\mathcal{T}$ that contain T pairwisely intersect in T and at most one additional vertex outside of T.

Figure 7.6: Illustration of the vertex set T (black vertices) and copies of H of the witness $\mathcal{T}$ (computed in line 3 of ReducedWitness) that contain T, assuming $h = 4$.

bounded easily. See Figure 7.6a for an illustration of the case $i = 0$. For $i > 0$, the set T contains less than $h-2$ vertices, and the copies of H whose vertex sets are supersets of T might also intersect outside of T, but then we can bound their number based on the induction hypothesis. See Figure 7.6b for an illustration of the case $i = 1$. Now, we show the induction in detail.

Let $i = 0$ and let T be a size-$(h-2)$ vertex subset such that $|\mathcal{C}| > 1 + hk$ (line 9), and let $\mathcal{C}' \subseteq \mathcal{C}$ be as in line 10. Clearly, $|\mathcal{C} \setminus \mathcal{C}'| = 1 + hk$. Since the copies of H in $\mathcal{C}$ pairwise intersect (due to the construction of $\mathcal{C}$ in line 8), at most one of them can be in $\mathcal{P}$. Let H_1 be that copy and assume that $H_1 \in \mathcal{C}'$. The remaining $k-1$ copies of H in $\mathcal{P} \setminus \{H_1\}$ can intersect with at most $h \cdot (k-1)$ copies of H in $\mathcal{C} \setminus \mathcal{C}'$, since the copies of H in $\mathcal{C}$ pairwise intersect exactly in T (because $\mathcal{W}$ is a maximal set of copies of H that pairwise intersect in at most $h-2$ vertices and $|T| = h-2$). As a consequence, there is at least one $H_2 \in \mathcal{C} \setminus \mathcal{C}'$ such that $V(H_2) \cap V(\mathcal{P}) = V(H_2) \cap V(H_1) = T$. We remove H_1 from $\mathcal{P}$ and add H_2 to it. As a consequence, $\mathcal{P}$ contains no copy of H from $\mathcal{C}'$.

For $i > 0$, let T be a size-$(h-2-i)$ vertex subset such that $|\mathcal{C}| > \sum_{t=0}^{i+1}(hk)^t$ (line 9), and let $\mathcal{C}' \subseteq \mathcal{C}$ be as in line 10. Again, we may assume that there exists an $H_1 \in \mathcal{P} \cap \mathcal{C}'$. We count the number of copies of H in $\mathcal{C} \setminus \mathcal{C}'$ that can intersect with $\mathcal{P} \setminus \{H_1\}$. Let $W := V(\mathcal{P} \setminus \{H_1\})$. Obviously, $|W| \leq h \cdot (k-1) < hk$. Each vertex $v \in W$ can "hit" at most $\sum_{t=0}^{i}(hk)^t$ copies of H in $\mathcal{C} \setminus \mathcal{C}'$, since by the induction hypothesis, there are at most $\sum_{t=0}^{i}(hk)^t$ copies of H whose vertex sets are supersets of $T \cup \{v\}$. Therefore, less than $hk \sum_{t=0}^{i}(hk)^t = \sum_{t=1}^{i+1}(hk)^t$ copies of H in $\mathcal{C} \setminus \mathcal{C}'$ intersect with $\mathcal{P} \setminus \{H_1\}$, and thus there is at least one left in order to replace H_1 (recall that $|\mathcal{C} \setminus \mathcal{C}'| = \sum_{t=0}^{i+1}(hk)^t$).

Thus, eventually we obtain a set $\mathcal{P}'$ of k vertex-disjoint copies of H such that $\mathcal{P}' \cap \mathcal{R} = \emptyset$. □

Lemma 7.9. *Algorithm* ReducedWitness *in Figure 7.5 runs in* $O(n^{3h} \cdot h^{h+2})$ *time.*

Proof. Since $|\mathcal{H}| \leq \binom{n}{h}$ there are $O(n^h)$ copies of H to consider. We first analyze the inner loop between lines 2–12. Analogously to Triangle Packing, line 3 can be executed in $O(n^h \cdot h)$ time by an iterative approach that adds an element H' from $\mathcal{H} \setminus (\mathcal{W} \cup \mathcal{R})$ to $\mathcal{W}$ if H' intersects with each element in $\mathcal{W}$ in at most $h-2$ vertices (the factor h is needed in order to check for two copies of H in how many elements they intersect). The for-loop in line 5 is executed exactly $h-2$ times. In each iteration of this loop, we iterate over all graphs in $\mathcal{W}$, which are at most $O(n^h)$ many. For each graph H' in $\mathcal{W}$, the algorithm tries subsets T of $V(H')$ (line 7), which takes $O(h^h)$ time. Then, constructing the set $\mathcal{C}$ for each subset T takes $O(n^h \cdot h)$ time (line 8). Testing the condition in line 9 can be done in $O(n^h)$ time, and the remaining tasks of choosing a subset $\mathcal{C}'$ of $\mathcal{C}$ in line 10, removing $\mathcal{C}'$ from $\mathcal{W}$, and adding $\mathcal{C}'$ to $\mathcal{R}$ take $O(n^h)$ time, since there are at most $O(n^h)$ copies of H to consider (and assuming that adding/removing an element to/from a set takes constant time). Summarizing, the algorithm needs $O(n^h \cdot h + h(n^h \cdot h^h \cdot n^h \cdot h)) = O(n^{2h} \cdot h^{h+2})$ time for each iteration of the repeat-until-loop, and this loop is executed at most $O(n^h)$ times, because $\mathcal{W}$ contains $O(n^h)$ graphs, and in each loop iteration except the last one, at least one graph is removed from $\mathcal{W}$. The total running time is therefore $O(n^{3h} \cdot h^{h+2})$. □

With the help of ReducedWitness we can state the following reduction rule. Recall that $\mathcal{H}$ is the set of all copies of H in G. Run ReducedWitness($\mathcal{H}$) to get a witness $\mathcal{W}$ and the set $\mathcal{R}$. All vertices that are only contained in copies of H in $\mathcal{R}$ are superfluous and can be removed:

Reduction Rule 7.5. *Replace* G *by* $G[V(\mathcal{H} \setminus \mathcal{R})]$.

Lemma 7.10. *Reduction Rule 7.5 is correct, that is,* G *has a size-*k H*-packing if and only if* $G[V(\mathcal{H} \setminus \mathcal{R})]$ *has a size-*k H*-packing.*

Proof. Obviously, a size-k H-packing in $G[V(\mathcal{H} \setminus \mathcal{R})]$ is also a size-k H-packing in G. If a size-k H-packing in G, which must be a subset of $\mathcal{H}$, contains copies of H in $\mathcal{R}$, then we can replace them by other copies not in $\mathcal{R}$ due to Lemma 7.8 and obtain a packing that does not contain any copy of H from $\mathcal{R}$. This packing is a size-k H-packing in $G[V(\mathcal{H} \setminus \mathcal{R})]$. □

In the following, we assume that the graph G is reduced with respect to Reduction Rule 7.4 and Reduction Rule 7.5, that $\mathcal{H}$ is the set of all copies of H in G, and that the set $\mathcal{W}$ of copies of H that pairwisely intersect in at most $h-2$ vertices is a witness with respect to $\mathcal{H}$. It remains to show how to bound the size of the witness $\mathcal{W}$ with respect to k; moreover, we have to bound the number of vertices outside of the witness. As mentioned before (see also Figure 7.4), the vertices outside of the witness form an independent set:

Lemma 7.11. *(1) The set $I := V \setminus V(\mathcal{W})$ forms an independent set in G and (2) each copy of H in $\mathcal{H} \setminus \mathcal{W}$ contains a vertex in I and $h-1$ vertices of some copy of H in $\mathcal{W}$.*

Proof. To proof works similar to the proof of Lemma 7.3. (1) If $G[I]$ contains an edge, which has to be part of some copy of H due to Reduction Rule 7.4, then at most $h-2$ vertices of that copy intersect with each $H \in \mathcal{W}$, contradicting the fact that $\mathcal{W}$ is maximal. (2) If a copy of H in $\mathcal{H} \setminus \mathcal{W}$ shares at most $h-2$ vertices with each copy of H in $\mathcal{W}$, then we again have a contradiction to the fact that $\mathcal{W}$ is maximal. □

It follows directly from Lemma 7.11 that each copy of H contains at most one vertex of I. Now, analogously to Triangle Packing, we bound the number of vertices in $V(\mathcal{W})$ and the number of copies of H in $\mathcal{W}$.

Lemma 7.12. *If $|V(\mathcal{W})| > 2h(hk)^{h-1}$ or if $|\mathcal{W}| > 2(hk)^{h-1}$, then G contains k vertex-disjoint copies of H.*

Proof. We use the same proof strategy as in the proof of Lemma 7.4. Assume that G does not contain a size-k H-packing. Let $\mathcal{P}$ be an H-packing of maximum size. Since $|\mathcal{P}| \leq k-1$, there are at most $h \cdot (k-1)$ vertices in $V(\mathcal{P})$. Each of these vertices is contained in at most

$$\sum_{t=0}^{h-2} (hk)^t$$

copies of H, since for $i = h-3$ each set T in line 7 of ReducedWitness (Figure 7.5) contains one vertex, and the number of copies of H containing T directly follows from the condition in line 9. We estimate the number of such copies of H as

$$\sum_{t=0}^{h-2} (hk)^t \leq 2(hk)^{h-2}.$$

This can be seen by assuming $h \geq 3$ and $k \geq 2$ (geometric series). These assumptions are correct, since for $h \leq 2$ H-Packing is polynomial-time solvable; moreover, $k = 1$ implies $\mathcal{P} = \emptyset$ (recall the assumption that G does not contain a size-k H-packing). As a consequence, the graph cannot contain any copy of H, thus $|\mathcal{W}| = 0$.

In summary, there are at most $h \cdot (k-1)$ vertices in $V(\mathcal{P})$, and each of them is contained in at most $2(hk)^{h-2}$ copies of H, hence we obtain $|\mathcal{W}| \leq 2(hk)^{h-1}$ and $|V(\mathcal{W})| \leq 2h(hk)^{h-1}$. □

It remains to bound the size of $I := V \setminus V(\mathcal{W})$. To this end, as for Triangle Packing, we define a bipartite auxiliary graph $G_{\mathcal{W}}$ as follows. The vertex set consists of I as one partite set and a set J as the other, where J contains a vertex v_X for each set $X \in \{V(H) \cap V(\mathcal{W}) \mid H \in (\mathcal{H} \setminus \mathcal{W})\}$ (that is, we have a

vertex for each possible intersection of the copies of H in $\mathcal{H} \setminus \mathcal{W}$ with the vertex set $V(\mathcal{W})$). For each $H \in (\mathcal{H} \setminus \mathcal{W})$ there is an edge between the vertex in the set $V(H) \cap I$ and the vertex v_X with $X := V(H) \cap V(\mathcal{W})$. Note that for each $H' \in \mathcal{W}$ there are at most h sets $X \subsetneq V(H')$ in $\{V(H) \cap V(\mathcal{W}) \mid H \in (\mathcal{H} \setminus \mathcal{W})\}$, and hence the size bound of $\mathcal{W}$ from Lemma 7.12 together with Lemma 7.11 part (2) yields $|J| < 2h(hk)^{h-1}$. The size of the independent set I then can be bounded exactly as for Triangle Packing with the following reduction rule; the proof of correctness is almost the same as the proof of Lemma 7.5 (replacing $G_{\mathcal{T}}$ with $G_{\mathcal{W}}$ and "triangle" with "copy of H").

Reduction Rule 7.6. *Compute a maximum matching in $G_{\mathcal{W}}$. Remove all unmatched vertices in I from G.*

The number of vertices in $I = V \setminus V(\mathcal{W})$ then can be bounded by the size of J, that is, there are at most $2h(hk)^{h-1}$ vertices of I remaining. Together with the at most $2h(hk)^{h-1}$ vertices in $V(\mathcal{W})$ (Lemma 7.12), in total there remain $O(k^{h-1})$ vertices.

Next, we show the running time. Reduction Rule 7.4 can be applied in $O(n^h)$ time (Lemma 7.7). Algorithm ReducedWitness takes $O(n^{3h})$ time (Lemma 7.9). Constructing the auxiliary graph $G_{\mathcal{W}}$ takes $O(n^h)$ time by iterating over all copies of H in $\mathcal{H} \setminus \mathcal{W}$. The graph $G_{\mathcal{W}}$ contains $O(n^{h-1})$ vertices and $O(n^h)$ edges (one for each $H \in (\mathcal{H} \setminus \mathcal{W})$, thus a maximum matching in $G_{\mathcal{W}}$ can be found in $O(n^h \cdot \sqrt{n^{h-1}})$ time [HK73]. Hence, the total running time of the kernelization algorithm is dominated by the running time $O(n^{3h} \cdot h^{h+2})$ of ReducedWitness. Recall that h denotes the number of vertices in H. We arrive at the following result.

Theorem 7.2. *H-Packing admits a problem kernel of $O(k^{|V(H)|-1})$ vertices, which can be constructed in $O(n^{3|V(H)|} \cdot |V(H)|^{|V(H)|+2})$ time.*

7.3 A New Problem Kernelization for Hitting Set

In this section, we adapt the kernelization algorithm for H-Packing given in Section 7.2 to obtain a kernelization algorithm for h-Hitting Set; the resulting problem kernel has some properties that make it possible to use the kernelization algorithm for many problems that can be reduced to h-Hitting Set.

h-Hitting Set
Input: A ground set V, a family $\mathcal{H}$ of subsets of V of size at most h, and an integer $k \geq 0$.
Question: Does there exist a subset $S \subseteq V$ of size at most k such that each subset in $\mathcal{H}$ contains at least one element from S?

The sets in $\mathcal{H}$ are also called *hyperedges*. In the following, we call the elements in V *vertices*. Let $V = V(\mathcal{H})$. The best-known problem kernel for h-Hitting

Set has $O(k^{h-1})$ vertices [Abu09]. This problem kernel can be used to obtain problem kernels for various graph problems that can be reduced to 3-Hitting Set [AF06, Abu09, DGH+09, HKMN09a]. The idea is to reduce the problem instance to 3-Hitting Set, to kernelize the 3-Hitting Set instance, and translate the resulting kernel back to the original problem. For general h-Hitting Set, the problem kernelization algorithm by Abu-Khzam [Abu09] replaces some hyperedges by smaller ones. Likewise, other problem kernels for h-Hitting Set replace some hyperedges by smaller ones [AF06, FG06, NR03]. This is a problem if one wants to kernelize a graph problem by reducing it to h-Hitting Set, kernelizing the resulting h-Hitting Set instance, and translating the h-Hitting Set kernel back to the original problem. In general, the resulting kernel for the original problem must be a valid input instance for the original problem, which cannot be guaranteed if the smaller hyperedges have no corresponding structure in the original problem.

Kratsch [Kra09] has adapted an $O(k^h)$-vertex problem kernel for h-Hitting Set by Flum and Grohe [FG06] such that it avoids replacing hyperedges by smaller ones; instead, it just removes some hyperedges that do not need to be considered. Kratsch [Kra09] uses this h-Hitting Set kernel to show that every problem in the classes MIN $F^+\Pi_1$ and MAX SNP (classes of constant-factor approximable problems) admits a polynomial-size problem kernel. Next, we show that with the technique for the H-Packing kernel in Section 7.2 one can obtain an $O(k^{h-1})$-vertex problem kernel such that no hyperedges are replaced by smaller ones. As an example problem to apply this kernel, we consider the following vertex deletion problem for hereditary graph properties (see also Chapter 6). Let $\mathcal{H}$ be a fixed set of graphs.

$\mathcal{H}$-Free Vertex Deletion
Input: An undirected graph $G = (V, E)$ and an integer $k \geq 0$.
Question: Does there exist a vertex subset $S \subseteq V$ of size at most k such that $G - S$ does not contain a (not necessarily induced) copy of some graph $H \in \mathcal{H}$?

The $\mathcal{H}$-Free Vertex Deletion problem is fixed-parameter tractable with respect to the parameter k [Cai96]. Obviously, it can be directly reduced to h-Hitting Set: simply enumerate all copies of H in G for all $H \in \mathcal{H}$ and use the vertex set of each copy as an hyperedge of the corresponding h-Hitting Set instance. One can use the kernelization algorithm for h-Hitting Set by Kratsch to obtain a $O(k^h)$-vertex problem kernel for $\mathcal{H}$-Free Vertex Deletion, where h is the largest number of vertices of a graph in $\mathcal{H}$.

In this section, we show that the technique used in Section 7.2 gives a problem kernel for h-Hitting Set of $O(k^{h-1})$ vertices that can be used to kernelize vertex-deletion problems for hereditary graph properties, in particular, $\mathcal{H}$-Free Vertex Deletion. The resulting kernelization works very similarly to the h-Hitting Set kernel by Abu-Khzam [Abu09]. The main difference is that it avoids introducing smaller hyperedges.

Definition 7.3. *We call a problem kernel* $(\mathcal{H}', k')$ *for a* h-Hitting Set *instance* $(\mathcal{H}, k)$ hyperedge-preserving*, if* $\mathcal{H}' \subseteq \mathcal{H}$.

Translating a vertex deletion problem such as $\mathcal{H}$-Free Vertex Deletion to h-Hitting Set, finding a hyperedge-preserving kernel, and removing from the original graph all vertices that are not contained in hyperedges of the h-Hitting Set kernel, directly yields a problem kernel for the corresponding vertex deletion problem. Note that the h-Hitting Set kernel by Kratsch [Kra09] is hyperedge-preserving, and this is the main property he needs for his kernelization results for problems in MIN $\mathrm{F}^{+}\Pi_1$ and MAX SNP.

In the following, we describe how to use the problem kernel from Section 7.2 to obtain a hyperedge-preserving problem kernel for h-Hitting Set. The description of the kernelization algorithm in Section 7.2 can be changed accordingly by replacing "copy of H" by "hyperedge". Note that the hyperedges must not necessarily have the same size. For an intuitive idea of the kernelization method, let $\mathcal{C}$ be a set of hyperedges that pairwise intersect in a vertex set T. If we further assume that the hyperedges in $\mathcal{C}$ pairwise intersect *exactly* in T, then it clearly holds that if $|\mathcal{C}| > k + 1$, then at least one vertex in T must be contained in a solution of size at most k in order to "hit" all hyperedges. However, unless the hyperedges have maximum size three, we do not know which vertex from T will be contained in a solution of size at most k (because $|T| \geq 2$), but we can remove "unnecessary" hyperedges: if we remove all but $k + 1$ hyperedges, then in the resulting reduced instance, a solution S of size at most k still contains a vertex from T; therefore, in the original instance, the removed hyperedges are still covered by S (and, of course, a solution for the original instance is also a solution for the reduced instance). We generalize this idea with the same algorithm as in Section 7.2.2 in order to derive a witness. In the following, we only sketch the proof modifications, showing that the approach in Section 7.2.2 also works for h-Hitting Set. In this new context, $\mathcal{R}$ is the set of all "unnecessary" hyperedges. In other words, $\mathcal{R}$ is the set of all removed hyperedges, that is, the union of all sets $\mathcal{C}'$ that are computed in the course of the execution of ReducedWitness.

Lemma 7.13 (Replacement for Lemma 7.8)**.** *There exists a hitting set* S *of size* k *of* $\mathcal{H}$ *if and only if there exists a hitting set* S' *of size* k *of* $\mathcal{H} \setminus \mathcal{R}$.

Proof. All line numbers refer to the algorithm ReducedWitness in Figure 7.5. For $i = 0$, let X be a size-$(h-2)$ vertex subset such that $|\mathcal{C}| > 1 + hk$ (line 9), and let $\mathcal{C}'$ be as in line 10. Since $1 + hk \geq k + 1$ for $h \geq 1$, and since the hyperedges in $\mathcal{C}$ pairwise intersect exactly in T, at least one vertex in T is in any optimal solution S. Otherwise, S would have to contain more than k vertices in order to contain at least one vertex of each hyperedge in $\mathcal{C}$. The same holds for $\mathcal{C} \setminus \mathcal{C}'$, since $|\mathcal{C} \setminus \mathcal{C}'| > k + 1$. Thus, even if we remove the hyperedges in $\mathcal{C}'$, we have $S \cap T \neq \emptyset$.

Analogous arguments can be applied for $i > 0$: let T be a size-$(h-2-i)$ vertex subset such that $|\mathcal{C}| > \sum_{t=0}^{i+1}(hk)^t$ (line 9) and let $\mathcal{C}' \subseteq \mathcal{C}$ be as in line 10.

Assume that an optimal solution S does not contain any vertex in T. Each vertex in $v \in S$ can "hit" at most $\sum_{t=0}^{i}(hk)^t + 1$ hyperedges in $\mathcal{C} \setminus \mathcal{C}'$, since by the induction hypothesis there are at most $\sum_{t=0}^{i}(hk)^t$ hyperedges that are supersets of $T \cup \{v\}$, and there might exist the hyperedge $T \cup \{v\}$. Therefore, since $k \leq (\sum_{t=0}^{i+1}(hk)^t)/h$ for $i > 0$ and since $h > 2$, at most

$$k \cdot (\sum_{t=0}^{i}(hk)^t + 1) = \frac{\sum_{t=0}^{i+1}(hk)^t}{h} + k < \sum_{t=0}^{i+1}(hk)^t$$

hyperedges of $\mathcal{C} \setminus \mathcal{C}'$ contain a vertex of S, a contradiction to the assumption that S is a solution. Therefore, $S \cap T \neq \emptyset$.

Recall that $\mathcal{R}$ is the set of all removed hyperedges, that is, the union of all sets $\mathcal{C}'$ that are computed in the course of the execution of ReducedWitness. Let S be a hitting set for $\mathcal{H}$. Obviously, S is also a hitting set for $\mathcal{H} \setminus \mathcal{R}$. Let S' be a hitting set for $\mathcal{H} \setminus \mathcal{R}$. Due to the above arguments, each hyperedge in $\mathcal{R}$ contains a vertex in S', and hence S' is also a hitting set for $\mathcal{H}$. □

Analogously to Reduction Rule 7.5, we can remove the hyperedges in $\mathcal{R}$ and work with the remaining instance. Let $\mathcal{H}$ be the remaining set of hyperedges and let $\mathcal{W}$ be a witness with respect to $\mathcal{H}$.

Lemma 7.14 (Replacement of Lemma 7.11)**.** *(1) The vertices $I := V(\mathcal{H}) \setminus V(\mathcal{W})$ form an independent set, that is, there are no hyperedges in $\mathcal{H}$ that contain two elements from I; (2) each hyperedge in $\mathcal{H} \setminus \mathcal{W}$ contains a vertex in I and $h-1$ vertices of some hyperedge in $\mathcal{W}$.*

The proof of this replacement lemma is completely analogous to the proof of Lemma 7.11. Then, the size of the witness can be bounded as follows.

Lemma 7.15 (Replacement of Lemma 7.12)**.** *If $|V(\mathcal{W})| > 2h(hk)^{h-1}$ or if $|\mathcal{W}| > 2(hk)^{h-1}$, then $(\mathcal{H}, k)$ is a no-instance.*

Proof. Assume that S is a solution for the h-Hitting Set instance $(\mathcal{H}, k)$. Since $|S| \leq k$ and each vertex $v \in S$ is contained in at most $2(hk)^{h-2}$ hyperedges (see proof of Lemma 7.12), $\mathcal{H}$ can contain at most $k \cdot 2(hk)^{h-2} < 2(hk)^{h-1}$ hyperedges and therefore $|V(\mathcal{H})| \leq 2h(hk)^{h-1}$. □

It remains to bound the size of I. The method is completely analogous to the method used for H-Packing. However, we need to show that this method is actually also correct for h-Hitting Set. We construct the graph $G_{\mathcal{W}}$ as described in Section 7.2.2 (replacing "copy of $\mathcal{H}$" by "hyperedge") and apply Reduction Rule 7.6, that is, we remove all unmatched vertices in I and all hyperedges in $\mathcal{H}$ that contain unmatched vertices. Let $\mathcal{H}'$ be the resulting h-Hitting Set instance.

Lemma 7.16. *The instance $(\mathcal{H}, k)$ is a yes-instance if and only if $(\mathcal{H}', k)$ is a yes-instance.*

Proof. ($\Rightarrow$) Let S be a size-k hitting set for $\mathcal{H}$. Then, obviously, $S \setminus I'$ is a hitting set for $\mathcal{H}'$ of size at most k.

($\Leftarrow$) Let M be the computed maximum matching in $G_{\mathcal{W}}$ and let I' be all unmatched vertices in I. Since M is maximum, $G_{\mathcal{W}}$ contains no M-augmenting path. The following definitions are as in the proof of Lemma 7.5. Let $I_1 \subseteq I \setminus I'$ be the set of vertices in $I \setminus I'$ to which there exists an M-alternating path from some vertex in I'. Each vertex $u \in I_1$ is an endpoint of an edge in M, because an M-alternating path from some vertex in I' to u begins with an unmatched vertex. Let $M' \subseteq M$ be the matching edges that have an endpoint in I_1, and let $J_1 := J \cap V(M')$ be the corresponding other endpoints of M'. Each edge in $G_{\mathcal{W}}$ corresponds to a hyperedge in $\mathcal{H} \setminus \mathcal{W}$. Thus, for each edge $\{v, v_X\}$ in M', where $v \in I$ and $v_X \in V(G_{\mathcal{W}}) \setminus I$, either u or some vertex in the size-$(h-1)$ subset $X \subsetneq V(G_{\mathcal{W}})$ corresponding to v_X must be in a hitting set. Given any hitting set S' for $\mathcal{H}'$, we can replace each vertex v in $S' \cap I_1$ by some vertex in $v_X \in J_1$, where $\{v, v_X\} \in M'$, yielding a set S'' of size $|S'|$. The set S'' is a hitting set for $\mathcal{H}'$: the set S'' contains at least one vertex from each vertex set X corresponding to $v_X \in J_1$. All hyperedges that contain a vertex from I_1 correspond to edges between I_1 and J_1 (due to the definition of I_1 and J_1). Hence, all hyperedges corresponding to edges between I_1 and J_1 contain a vertex from S''. All hyperedges in $\mathcal{H} \setminus \mathcal{H}'$ correspond to edges between I' and J_1; therefore, all hyperedges in $\mathcal{H} \setminus \mathcal{H}'$ contain a vertex in S''. This shows that S'' is a hitting set for $\mathcal{H}$. □

Using this lemma, the size of I can be bounded by $2h(hk)^{h-1}$. The running time of the whole kernelization algorithm is again dominated by the running time of ReducedWitness, which is $O(|\mathcal{H}|^3 + h^{h+2})$ (using an analogous analysis as in the proof of Lemma 7.9). Therefore, we obtain the following.

Theorem 7.3. *h-Hitting Set admits a hyperedge-preserving problem kernel of $O(k^{h-1})$ vertices, which can be constructed in $O(|\mathcal{H}|^3 \cdot h^{h+2})$ time.*

From this theorem, a problem kernel for $\mathcal{H}$-Free Vertex Deletion immediately follows.

Corollary 7.1. *$\mathcal{H}$-Free Vertex Deletion admits an $O(k^{h-1})$-vertex problem kernel, which can be constructed in $O(n^{3h} \cdot h^{h+2})$ time, where h is the number of vertices of a largest graph $H \in \mathcal{H}$.*

7.4 Outlook

Our approach for general H-Packing probably would also work for Set Packing. However, this would only give a better kernel with respect to the number of elements, not with respect to the number of sets. Therefore, it would not improve the known kernelization results [FKN+07] (where the kernel size is expressed with

respect to the number of sets rather than with respect to the elements). Abu-Khzam has given such a kernel with an improved number of elements compared to [FKN+07] for 3-Set Packing [AK09]. One of the key ingredients for obtaining a problem kernel with $O(k^{|V(H)|-1})$ vertices instead of $O(k^{|V(H)|})$ vertices for H-Packing is the matching technique to bound the number of vertices in the remaining independent set. It would be interesting to know whether it is possible to bound the size of structures different from independent sets by similar techniques In this way, a witness with less vertices and edges could be possible, which could make a better kernel size (compared to the results in this chapter) possible.

One of the next steps in research could be to develop parameterized (kernelization) algorithms that are especially tailored for packing-like problems in computational biology [ABWB+09]. Such algorithms could also be implemented and tested on real-world instances; in particular, using the color-coding technique, which has already shown its practical potential (cf. [Hüf07]) and which can also be combined with other techniques like divide and conquer [CKL+09], could be a good first approach for algorithm engineering on packing problems.

Chapter 8

Induced Matching

A well-studied generalization of the classical Maximum Matching problem is Distance-d Matching: given an undirected graph and an integer k, the task is to find at least k edges that have pairwise distance at least d. Clearly, for $d = 1$, this problem is exactly Maximum Matching and therefore polynomial-time solvable. For $d > 1$, the problem is NP-complete [SV82]. In this chapter, we concentrate on Distance-d Matching for $d = 2$, which is also known as Induced Matching. While there exists a considerable number of results in the context of classical complexity theory, there are only very few results concerning the parameterized complexity of Induced Matching. In general, the problem is W[1]-hard. In this chapter, we settle the parameterized complexity of Induced Matching on several classes of graphs where it remains NP-complete, including planar graphs, graphs of bounded degree, bipartite graphs, graphs with girth at least six, line graphs, and graphs of bounded treewidth.

8.1 Introduction and Known Results

An *induced matching* M of a graph $G = (V, E)$ is an edge-subset $M \subseteq E$ such that M is a matching and no two edges of M are joined by an edge of G, that is, the edges in M have distance at least two. In other words, the set of edges of the subgraph induced by $V(M)$ is precisely the set M. The decision version of Induced Matching is defined as follows.

Induced Matching
Input: An undirected graph $G = (V, E)$ and an integer $k \geq 0$.
Question: Does G have an induced matching of at least k edges?

Induced Matching, also called Strong Matching, was introduced as a variant of the maximum matching problem with a distance constraint by Stockmeyer and Vazirani [SV82] as the "risk-free" marriage problem.[1] Note that In-

[1]Decide whether there exist at least k pairs such that each married person is compatible with no married person except the one he or she is married to.

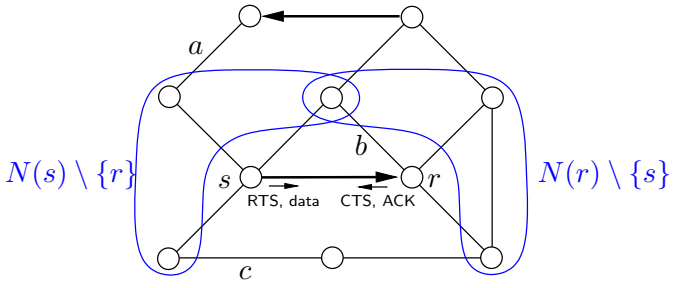


Figure 8.1: Example of a graph that models a wireless communication network with two simultaneous transmissions, which are denoted by the bold edges with arrows. The corresponding edges form an induced matching in the graph. Note that this is not an induced matching of maximum size; a larger induced matching is formed by the edges a, b, and c.

duced Matching is equivalent to 1-Regular Induced Subgraph (see Chapter 5).

Applications. Induced Matching has natural applications in (wireless) communication networks. In the following, we briefly outline an application occurring in wireless networks that are based on the widely used IEEE 802.11 standard [IEE07]. One difficulty in a wireless network is that the communication nodes (e.g., laptops, mobile phones, wireless access points) share the same communication medium (e.g., a certain channel that is determined by the frequency of the underlying carrier signal). A network node cannot send and receive at the same time on the same medium, and it can only receive a signal from another node if that node is the only node within range that is sending. We can model a wireless communication network using an undirected graph, where the vertices are the communication nodes, and two vertices are adjacent if the corresponding nodes are within radio range. In such a network, the communication between the nodes has to be controlled in order to avoid interference. A media access (MAC) protocol, which is implemented in each node, provides such kind of control. A very popular MAC protocol is the IEEE 802.11 standard [IEE07]. It uses an RTS/CTS control scheme (see also the example in Figure 8.1): if a node s (sender) wants to send data to an adjacent node r (receiver), then s broadcasts a request-to-send (RTS) message addressed to r (but which is received by all neighbors of s). After that, if r is not aware of any other ongoing communication, then it sends a clear-to-send (CTS) message addressed to s. After hearing the CTS, s starts sending the message containing the data. If r receives that data successfully, it responses with an acknowledgement (ACK) message, terminating the transmission. If s does not hear an ACK after sending the data, it retries to send the data a prespecified amount of times. Note that some vertex in $N(r) \setminus N(s)$ could receive data from other nodes in the network while s is sending. However,

in order to receive data, such a vertex would have to follow the protocol and send a CTS message, which would reach r and interfere the transmission from s to r. Hence, while r and s are communicating, the nodes in $N(\{r, s\})$ (which are aware of the transmission between r and s due to the RTS and/or CTS messages) do not send an RTS nor respond to other RTS requests from other nodes in the network. As a direct consequence the simultaneous transmissions in the network correspond to an induced matching in the network graph. The *network capacity*, which is the maximum possible number of simultaneous transmission in the network, is therefore exactly the size of a maximum induced matching in the corresponding graph.

There are several papers studying the IEEE 802.11 standard with respect to Induced Matching, also including experimental results [BBK+04, SMS06, SSM06]. Finding a large induced matching is an important task when solving *packet routing* and *packet scheduling* problems under the above model [KMPS04, SMS06, KMPS07]. Here, the task is to route packets through the network with the constraint that the simultaneous transmissions must always form an induced matching.

Wireless networks often have a small treewidth, which can be exploited for practical algorithms [Kos99]. Induced Matching has been studied in the context of the IEEE 802.11 standard for planar graphs [BBK+04]. This, in particular, motivates our studies of planar graphs and graphs of bounded treewidth.

Other applications of Induced Matching exist for secure communication channels, VLSI and network flow problems [GL00], and a single frequency allocation problem in communication networks that are similar to the above IEEE 802.11 wireless networks [Rei04]. There exist also applications for Distance-d Matching where the assumption is that two nodes must be within distance at least d in order to avoid interference [SMS06, SSM06].

Hardness and Approximation. The studies of Induced Matching were initiated by Stockmeyer and Vazirani [SV82] by showing it to be NP-complete on general graphs. Concerning special graph classes, it is known that it is NP-complete on planar graphs of maximum degree four [KS03], bipartite graphs of maximum degree three, C_4-free bipartite graphs [Loz02], r-regular graphs for $r \geq 5$, line graphs, chair-free and claw-free graphs, Hamiltonian graphs [KR03], and unit disk graphs [BBK+04].[2]

Regarding polynomial-time approximability, in general graphs the problem cannot be approximated to within a factor of $n^{1/2-\epsilon}$ for any $\epsilon > 0$ unless P = NP [OFGZ08]. It is known that Induced Matching is APX-complete on r-regular graphs, for all $r \geq 3$, and bipartite graphs with maximum degree three [DMZ05]. Moreover, for r-regular graphs it is NP-hard to approximate Induced Matching to within a factor of $r/2^{O(\sqrt{\ln r})}$ [CC04]. There exists an

[2]For the graph classes that are not directly used in this thesis, see the book by Brandstädt et al. [BLS99] for the corresponding definitions.

approximation algorithm for the problem on r-regular graphs ($r \geq 3$) with asymptotic performance ratio $r - 1$ [DMZ05]; this has subsequently been improved to $0.75r + 0.15$ [GL05]. Moreover, there exists a polynomial-time approximation scheme (PTAS) for planar graphs of maximum degree three [DMZ05] and disk graphs [BBK$^+$04].

Polynomial-Time Solvable Cases. Induced Matching is known to be linear-time solvable for trees [Zit99], and polynomial-time solvable for chordal graphs [Cam89], weakly chordal graphs [CST03], circular arc graphs [GL93], trapezoid graphs, interval-dimension graphs, and comparability graphs [GL00], LAT-free graphs [Cha03], interval-filament graphs, polygon-circle graphs, AT-free graphs [Cam04], (P_5,D_m)-free graphs [KR03, LR03], (P_k,$K_{1,n}$)-free graphs [LR03], (bull, chair)-free graphs, line graphs of Hamiltonian graphs [KR03], and graphs where the maximum matching and the maximum induced matching have the same size [KR03]. For graphs in which the maximum matching and maximum induced matching have the same size, Cameron and Walker [CW05] provide a simple characterization of these graphs, which also leads to a simpler recognition algorithm for such graphs.

Parameterized Complexity. In contrast to the above results, little is known about the parameterized complexity of Induced Matching. First of all, since Induced Matching is equivalent to 1-Regular Induced Subgraph (Chapter 5), it is W[1]-hard with respect to the parameter k denoting the size of the induced matching (Theorem 5.4). Therefore, it is of interest to study the parameterized complexity of the problem in those restricted graph classes where it remains NP-complete. Faudree et al. [FGST89] showed that bipartite graphs of maximum degree d without an induced matching of size $k+1$ have at most kd^2 edges. This immediately gives a problem kernel of at most kd^2 edges in bipartite graphs with maximum degree d, showing that Induced Matching is fixed-parameter tractable on such graphs with respect to the parameter k for any constant d. Apart from that, to the best of our knowledge, our results presented in Section 8.2 first settled the parameterized complexity of the NP-complete Induced Matching on planar graphs, graphs of bounded degree, bipartite graphs, graphs with girth at least six, line graphs, and graphs of bounded treewidth. Meanwhile, our linear kernel for Induced Matching on planar graphs presented in Section 8.2.2 has been improved by Kanj et al. [KPXS09] and Kowalik et al. [KWKE09]. Kanj et al. [KPXS09] show that Induced Matching on planar graphs admits a problem kernel of at most $40k$ vertices and give two fixed-parameter algorithms, one with running time $O(2^{159\sqrt{k}} + n)$, and one with running time $O(91^k + n)$, which is faster than the first algorithm for small values of k. Kowalik et al [KWKE09] improve these results to a problem kernel of at most $28k$ vertices and a fixed-parameter algorithm with running time $O(2^{26\sqrt{k}} + n)$. Moreover, both Kanj et al. [KPXS09] and Kowalik et al. [KWKE09] give corresponding results for graphs

of bounded genus. While our kernelization result on planar graphs is based on a region decomposition technique, the results by Kanj et al. [KPXS09] and Kowalik et al. [KWKE09] are based on a study of the combinatorial properties of *twinless* graphs (two vertices u, v are *twins* if $N(u) = N(v)$). The region decomposition technique is a quite universal approach that can be used to kernelize problems on planar graphs (see Section 8.2.2); in this thesis, we concentrate on the main ideas of how to apply the decomposition technique for Induced Matching and omit some details.

In the next section, we give our results concerning the fixed-parameter tractability of Induced Matching in several graph classes.

8.2 Parameterized Complexity of Induced Matching on Restricted Graph Classes

In this section, we give linear problem kernels for planar graphs and bounded-degree graphs. For graphs of girth at least six, which also include C_4-free bipartite graphs, we can show a simple kernel with a cubic number of vertices (that is, $O(k^3)$ vertices). Moreover, we show that Induced Matching is fixed-parameter tractable for line graphs with respect to the parameter k. Finally, we give an algorithm for graphs of bounded treewidth using an improved dynamic programming approach, which runs in $O(4^\omega \cdot n)$ time, where ω is the width of the given tree decomposition. This extends an algorithm for Induced Matching on trees by Zito [Zit99]. On the negative side, we show that Induced Matching remains $W[1]$-hard on bipartite graphs. See Table 8.1 for an overview of these results, also including the results by Kanj et al. [KPXS09].

In the next section, we give the results for bounded-degree graphs, graphs of girth at least six, bipartite graphs, and line graphs. These results are simple and meant to provide some first-time insight into the parameterized complexity of Induced Matching on graphs in these classes. We then give a linear problem kernel on planar graphs in Section 8.2.2. Since this result has been improved by Kanj et al. [KPXS09], we give only a short excerpt of the proof in order to illustrate the main ideas; the missing details can be found in our journal version [MS09b]. Finally, we give the dynamic programming algorithm for graphs of bounded treewidth in Section 8.2.3.

8.2.1 Basic Results

The following results are basic first-time fixed-parameter tractability results for several graph classes where Induced Matching remains NP-hard.

Bounded-Degree Graphs. We show that Induced Matching admits a linear problem kernel on graphs of maximum degree d for some constant d.

Table 8.1: Parameterized complexity results for NP-complete variants of Induced Matching. Here, k denotes the size of the induced matching, and ω denotes the treewidth of the input graph.

graph class	par.	result	reference
general	k	$W[1]$-hard	Sec. 5.4
bipartite	k	$W[1]$-hard	Sec. 8.2.1
bounded degree	k	$O(k)$ kernel	Sec. 8.2.1
graphs of girth ≥ 6	k	$O(k^3)$ kernel	Sec. 8.2.1
line graphs	k	$O(2.448^{3k} \cdot k^{5.5} + k^2 n^2)$ alg.	Sec. 8.2.1
planar	k	$O(k)$ kernel	Sec. 8.2.2
		$28k$ kernel	[KWKE09]
		$O(2^{26\sqrt{k}} + n)$ alg.	[KWKE09]
bounded genus g	k	$(49 + 7\sqrt{1+48g})k/2 + 10g - 9$ kernel	[KWKE09]
		$O(\binom{7+\sqrt{1+48g)k}}{2k} k^2 + n)$ alg.	[KPXS09]
bounded treewidth	ω	$O(4^\omega \cdot n)$ alg.	Sec. 8.2.3

Proposition 8.1. *On graphs of maximum degree d,* Induced Matching *admits a problem kernel of $O(k \cdot d^2)$ vertices. The kernel can be constructed in $O(n)$ time for constant d.*

Proof. Let G be a graph with maximum degree d, where d is some constant. Let M be any maximal induced matching of G found by the following greedy algorithm. The algorithm repeatedly selects an arbitrary edge e, adds it to the solution, and deletes $N[e]$. This process is repeated until no more edges remain. Since the degree of the graph is bounded by d, selecting an edge and deleting its closed neighborhood takes constant time only, and the process is repeated at most $\lfloor n/2 \rfloor$ times, thus the whole greedy algorithm runs in $O(n)$ time.

If $|M| \geq k$, then we have found a solution and return "yes-instance". Therefore, assume that $|M| < k$. Define S_1 and S_2 by $S_1 := N(V(M))$ and $S_2 := N(S_1) \setminus V(M)$. Note that all neighbors of vertices in S_2 are in the set S_1 since if a vertex $u \in S_2$ has a neighbor $v \notin S_1$, then $\{u, v\}$ could be added to the induced matching, contradicting its maximality. Clearly, $|S_1| < 2kd$ and $|S_2| < 2kd^2$. Since $V(G) = V(M) \cup S_1 \cup S_2$, it immediately follows that $|V(G)| < 2k(1 + d + d^2)$. □

Graphs Without Small Cycles. Induced Matching is NP-hard on C_4-free bipartite graphs [Loz02]. Since the class of C_4-free bipartite graphs is properly contained in the class of graphs with girth at least six (that is, there are no cycles of length five or smaller), Induced Matching is NP-hard on the latter graph class as well.

Proposition 8.2. *On graphs with girth at least six,* Induced Matching *admits*

a problem kernel of $O(k^3)$ vertices. The kernel can be constructed in $O(n+m)$ time.

Proof. Let G be a graph with girth at least six. If a vertex has more than one degree-one neighbor, remove all but one of these neighbors. Repeat this until no longer possible. If every vertex has degree at most k, then we obtain a kernel of $O(k^3)$ vertices immediately from Proposition 8.1. Therefore, assume that there exists a vertex u of degree at least $k+1$. Let $S := \{v_1, \ldots, v_{k+1}\}$ be a set of $k+1$ neighbors of u. Since G has no cycles of length three (a shortest cycle has length at least six), S is independent. At most one vertex of S has degree one. Assume without loss of generality that the vertices in $\{v_1, \ldots, v_k\}$ have degree at least two. For $1 \leq i < j \leq k$, v_i and v_j do not have any common neighbors as otherwise we obtain a cycle of length four. For $1 \leq i \leq k$, let z_i be a neighbor of v_i. Again $\{z_1, \ldots, z_k\}$ must form an independent set as otherwise we obtain a cycle of length five. But then $\{(v_1, z_1), \ldots, (v_k, z_k)\}$ is an induced matching of size k. Therefore, we can either find an induced matching of size at least k in $O(n+m)$ time (removing the degree-one neighbors can be done in linear time, and computing $N(u)$ and its neighborhood can also be done in linear time) or obtain a kernel of size $O(k^3)$. □

Hence, Induced Matching, which is W[1]-hard with respect to the parameter k in general graphs, becomes fixed-parameter tractable for graphs without short cycles. This effect, that is, that sometimes short cycles make a problem W-hard, has also been observed by Raman and Saurabh [RS08] for various important problems in parameterized complexity, including Dominating Set and Independent Set.

Bipartite Graphs. For bipartite graphs we show that Induced Matching is W[1]-hard. We give a reduction from the W[1]-complete Irredundant Set problem [DFR00]. A *private neighbor* of a vertex $u \in V'$ is a vertex $u' \in N[u]$ (possibly $u' = u$) such that for every vertex $v \in V' \setminus \{u\}, u' \notin N[v]$.

Irredundant Set
Input: An undirected graph $G = (V, E)$ and an integer $k \geq 0$.
Question: Does there exist a set $V' \subseteq V$ of size at least k where each vertex $u \in V'$ has a private neighbor?

Proposition 8.3. *In bipartite graphs,* Induced Matching *is W[1]-hard with respect to the parameter k.*

Proof. We prove the proposition by a parameterized reduction from Irredundant Set. Let (G, k) be an instance of Irredundant Set. Construct a bipartite graph G' as follows. Construct two copies of the vertex set of G and call these V' and V''; the copies of a vertex $u \in V(G)$ from V' and V'' are denoted as u' and u'', respectively. Define $V(G') = V' \cup V''$ and $E(G') = \{\{u', u''\} : u \in$

$V(G)\} \cup \{\{u', v''\}, \{v', u''\} : \{u, v\} \in E(G)\}$. We show that the graph G has an irredundant set of size k if and only if G' has an induced matching of size k.

($\Rightarrow$) Suppose $S = \{w_1, \ldots, w_k\} \subseteq V(G)$ is an irredundant set of size k in G. For $1 \leq i \leq k$, let x_i be the private neighbor of w_i. Then, for all i, $\{w_i', x_i''\}$ is an edge in G'. Since the x_i's are private neighbors there is no edge $\{w_j, x_i\}$ in G for all $j \neq i$ and therefore no edge $\{w_j', x_i''\}$ in G'. As a consequence, the edges $\{w_1', x_1''\}, \ldots, \{w_k', x_k''\}$ form an induced matching in G'.

($\Leftarrow$) If $M = \{e_1, \ldots, e_k\}$ is an induced matching in G' of size k, then for each $e_i = \{u_i', v_i''\}$ there is no edge $e_j = \{u_j', v_j''\}$, $j \neq i$, such that u_j' and v_i'' are adjacent in G', that is, v_i is a private neighbor of u_i in G. Therefore, the vertices $u_1, \ldots, u_k$ form an irredundant set in G. This completes the proof. □

Line Graphs. The line graph $L(G)$ of a graph G is defined as follows: the vertex set of $L(G)$ is the edge set of G; two "vertices" e_1 and e_2 of $L(G)$ are connected by an edge if e_1 and e_2 share an endpoint. More formally, we have

$$L(G) := (E(G), \{\{e_1, e_2\} : e_1, e_2 \in E(G) \wedge e_1 \cap e_2 \neq \emptyset\}).$$

A graph H is a line graph if there exists a graph G such that $H = L(G)$. It is well-known (see, e.g., [FFR97]) that if H is a line graph, then it does not contain any induced $K_{1,3}$ (also known as *claw*). Induced Matching is NP-complete on line graphs (and hence claw-free graphs) [KR03]. Given an undirected graph H, it is possible to test in time $\max\{|V(H)|, |E(H)|\}$ (that is, linear time) whether H is a line graph and if so construct G such that $H = L(G)$ [Rou73].

Lemma 8.1. *Let H be a line graph and let $H = L(G)$. Then H has an induced matching of size at least k if and only if G has at least k vertex-disjoint copies (not necessarily induced) of a P_3, the path on three vertices.*

Proof. Let $\{e_1, \ldots, e_k\}$ be an induced matching of size k in H. From the definition of a line graph it follows that each edge e_i corresponds to a path $p_i = (x_i, y_i, z_i)$ in the graph G. The set $\bigcup_{i=1,\ldots,k}\{x_i, y_i, z_i\}$ contains exactly $3k$ vertices. Moreover, the sets $\{x_i, y_i, z_i\}$ and $\{x_j, y_j, z_j\}$ are disjoint for $i \neq j$: if any two vertices, one from path p_i and the other from p_j, are identical, then an endpoint of e_i would be connected to an endpoint of e_j, contradicting that e_i and e_j are part of an induced matching. This shows that G contains k vertex-disjoint copies of P_3. Conversely, if G has k vertex-disjoint copies of P_3, then the edges corresponding to these paths form an induced matching in H. □

The problem of checking whether a given graph G has k copies of P_3 can be solved in $O(2.448^{3k} \cdot k^{5.5} + k^2 n^2)$ time [FR09].

Proposition 8.4. *On line graphs,* Induced Matching *can be solved in $O(2.448^{3k} \cdot k^{5.5} + k^2 n^2)$ time and is therefore fixed-parameter tractable with respect to the parameter k.*

8.2.2 A Linear Problem Kernel on Planar Graphs

The linear kernel for Induced Matching on planar graphs is based on a kernelization technique first introduced by Alber et al. [AFN04] to show that Dominating Set admits a linear kernel on planar graphs. The result for the kernel size has subsequently been improved by Chen et al. [CFKX07], and they also show lower bounds on the kernel size for Dominating Set, Vertex Cover, and Independent Set on planar graphs. The technique developed by Alber et al. [AFN04] combined with problem-specific data reduction rules has been adapted for Full-Degree Spanning Tree [GNW09] and Planar Connected Dominating Set [LMS09]. Moreover, Fomin and Thilikos [FT04] extend the technique to graphs of bounded genus. Guo and Niedermeier [GN07b] give a generic kernelization framework for NP-hard problems on planar graphs based on that technique. Bodlaender et al. [BFL$^+$] give a tool to easily determine whether a problem admits a polynomial or linear kernel on planar graphs. They show that all problems expressible in Counting Monadic Second Order Logic [Cou90] that fulfill certain properties, admit a polynomial or linear kernel (depending on the properties) on graphs of bounded genus (and thus also for planar graphs). However, due to the generality of this approach, the hidden constants are huge; for a "small" problem kernel that is tailored for the individual problem at hand, using the framework by Guo and Niedermeier [GN07b] seems more appropriate.

For our kernel, we basically adapt and extend the technique introduced by Alber et al. [AFN04]; however, we base our proof on *edge* subset solutions rather than vertex subsets as in other applications of that technique, because it seems more appropriate for the particular case of Induced Matching. As pointed out earlier, our kernelization result presented in this section has been improved by Kanj et al. [KPXS09]. They use a completely different approach, not based on the general kernelization technique. In essence, they show that one can use a maximal matching in the graph for analyzing the kernel size; basically, if the graph is reduced with respect to a reduction rule that removes all twins from a graph, then not too many edges in this maximal matching can be pairwise adjacent if the graph is planar. Their kernel can be obtained in linear time as well, thus their approach is clearly the method of choice for deriving a good problem kernel for Induced Matching on planar graphs. However, their approach is not as generally applicable as the kernelization technique used for our kernelization result. For instance, it seems to be difficult to adapt their technique for kernelizing Distance-d Matching, that is, the problem of finding at least k edges that have pairwise distance at least d for some constant d. With the technique used in our proof, showing a kernel for Distance-d Matching is still a technically demanding task, but it seems to be in closer reach. For this reason, we give a short description of the linear kernel for Induced Matching on planar graphs based on the general kernelization technique, concentrating on the central ideas and omitting many details.

Data Reduction Rules

In order to show our kernel, we employ the following data reduction rules. These rules stem from the simple observation that if two vertices have the same neighborhood, one of them can be removed without affecting the size of a maximum induced matching. Compared to the data reduction rules applied in other proofs of planar kernels [AFN04, CFKX07, GNW09], these data reduction rules are quite simple and can be carried out in $O(n+m)$ time on general graphs (and hence in linear time on planar graphs).

Reduction Rule 8.1. *Delete vertices of degree zero.*

Reduction Rule 8.2. *If a vertex u has two neighbors x, y of degree 1, then delete x.*

Reduction Rule 8.3. *If two there are two degree-two vertices x and y such that $N(x) = N(y) = \{u, v\}$, then delete x.*

Note that these data reduction rules are parameter-independent. The reduction rule used by Kanj et al. [KPXS09], which removes all twins from the graph (recall that two vertices u, v are twins if $N(u) = N(v)$), can be considered as a generalization of our rules, since our rules remove twins with at most two neighbors.

Lemma 8.2. *Reduction Rules 8.1, 8.2, and 8.3 are correct.*

Proof. Obviously, none of these rules destroys planarity. The correctness of Reduction Rule 8.1 is obvious, since no isolated vertex can be part of an edge. Concerning Reduction Rule 8.2, observe that only one edge incident to u can be part of an induced matching. The correctness of Reduction Rule 8.3 can be seen as follows. Let G be a graph and M an induced matching for G. If one of the vertices x or y is an endpoint of an edge in M, then either u or v is the other endpoint of that edge since x and y have no other neighbors. Suppose, without loss of generality, that $\{u, x\}$ is a matching edge. Since u and y are adjacent, y cannot be an endpoint of an edge in M, and since x is adjacent to v, v cannot be an endpoint of an edge in M. For this reason, we can get a new matching $M' := (M \setminus \{\{u, x\}\}) \cup \{\{u, y\}\}$, which has the same size as M and is still induced, and it is an induced matching for $G' := G - x$. The case where no vertex in $\{x, y\}$ is an endpoint of an edge in M is obvious. The reverse direction is trivial, as any induced matching M' for G' is also an induced matching for G. □

Lemma 8.3. *Reduction Rules 8.1, 8.2, and 8.3 can be exhaustively applied in linear time on planar graphs as well as on general graphs.*

Proof. We first delete all isolated vertices in $O(n)$ time in order to reduce the graph with respect to Reduction Rule 8.1. Then, we apply Reduction Rule 8.3. For each vertex u of the graph we check which neighbors of u can be deleted.

To this end, we determine in $O(\deg(u))$ time all degree-two neighbors of u; then we group together all such neighbors whose second neighbor is the same. For each group, we mark all but one vertex for deletion. After having done this for every vertex we delete the marked vertices. Finally, we apply Reduction Rule 8.2. For each vertex u we determine in $O(\deg(u))$ time all degree-one neighbors of u, and delete all but one. The running time to exhaustively apply each rule is $O(\sum_{u \in V}(1 + \deg(u)))$, which is bounded by $O(n + m)$.

It remains to explain why we need to check every vertex for each rule only once, and why we first apply Reduction Rule 8.3 and then Reduction Rule 8.2. It is easy to verify that for each rule the following holds: a vertex that is not deleted during the application of the rule does not become a candidate for deletion with respect to the rule *after* the application of that rule on other vertices. Moreover, we have to justify why we apply Reduction Rule 8.3 before Reduction Rule 8.2. If Reduction Rule 8.3 cannot be applied anymore, then the application of Reduction Rule 8.2 cannot cause any situation where Reduction Rule 8.3 could be applied again. This does not hold if we apply the rules the other way around. The application of Reduction Rule 8.1 at the beginning is obviously correct. □

The following theorem is our main result. Its proof spans the remainder of this section.

Theorem 8.1. *On planar graphs,* Induced Matching *admits a linear problem kernel with respect to the parameter solution size k. The kernel can be constructed in linear time.*

Let $G = (V, E)$ be a planar graph reduced with respect to Reduction Rules 8.1, 8.2, and 8.3, for which any induced matching contains at k edges. We show that then $|V| = O(k)$. This result together with Lemma 8.2 and 8.3 shows Theorem 8.1.

For the proof, we assume to be given a maximum induced matching M of size k of G. The general strategy is to show that either $|V| = O(k)$ holds or that M cannot be of maximum size. The basic observation is that if M is a maximum induced matching of $G = (V, E)$, then for each vertex $v \in V$ there exists a vertex $u \in V(M)$ such that $d(u, v) \leq 2$. Otherwise, we could add an edge to M and obtain a larger induced matching. Then, since every vertex in the graph is within distance at most two to some vertex in $V(M)$, we know, roughly speaking, that each edge in M is within distance at most four to at least one other edge in M. This leads to the idea of regions "in between" matching edges that are close to each other. We will see that these regions cannot be too large if the graph is reduced with respect to the above data reduction rules. Moreover, we show that there cannot be many vertices that are not contained within such regions.

This idea of a region decomposition was introduced in [AFN04], but the definition of a region as it appears there is much simpler since the regions are defined between vertices, and they are smaller. The remaining part of this section is

dedicated to the proof of Theorem 8.1. First, we define a "maximal region decomposition" of a reduced graph that contains only $O(|M|)$ regions. Then, we outline the proof that each region only contains a constant number of vertices. In this part lies the most important difference to the kernel by Kanj et al. [KPXS09]; as mentioned before, they use a more general reduction rule that generates a twinless graph, and then their proof is based on combinatorial arguments for twinless graphs. In our proof, we divide the vertices inside a region into layers, which facilitates the proof of the size bound. It seems plausible that this will also help in analyzing the size of regions for other problems like Distance-d Matching with higher distance constraints. Here, we give the full proofs of the most important building blocks that are needed for the layer-wise analysis of the region size, but we only sketch the ideas of how to apply them. Concerning the number of vertices outside of regions, we only give a few comments on how to bound them. Recall that this section only aims to give an idea of how to prove the kernel size; all missing parts can be found in our paper [MS08a].

Maximal Region Decomposition

The basic concepts used for the kernel size proof are defined in the following.

Definition 8.1. *Let G be a plane graph and let M be a maximum induced matching of G. For edges $e_1, e_2 \in M$, a* region $R(e_1, e_2)$ *is a closed subset of the plane such that*

1. *the boundary of $R(e_1, e_2)$ is formed by two length-at-most-four paths*
   - *$(a_1, \ldots, a_2)$, $a_1 \neq a_2$, between $a_1 \in e_1$ and $a_2 \in e_2$,*
   - *$(b_1, \ldots, b_2)$, $b_1 \neq b_2$, between $b_1 \in e_1$ and $b_2 \in e_2$, and*

   by e_1 if $a_1 \neq b_1$ and e_2 if $a_2 \neq b_2$;
2. *for each vertex x in the region $R(e_1, e_2)$, there exists a vertex $y \in V(\{e_1, e_2\})$ such that $d(x, y) \leq 2$;*
3. *no vertices inside the region other than the endpoints of e_1 and e_2 are from M.*

The set of boundary vertices of R is denoted by δR. We write $V(R(e_1, e_2))$ to denote the set of vertices of a region $R(e_1, e_2)$, that is, all vertices strictly inside $R(e_1, e_2)$ together with the boundary vertices δR. A vertex in $V(R(e_1, e_2))$ is inside R.

Note that the two enclosing paths may be identical; the corresponding region then consists solely of a simple path of length at most four. Note also that e_1 and e_2 may be identical. For an example of a region see Figure 8.2a.

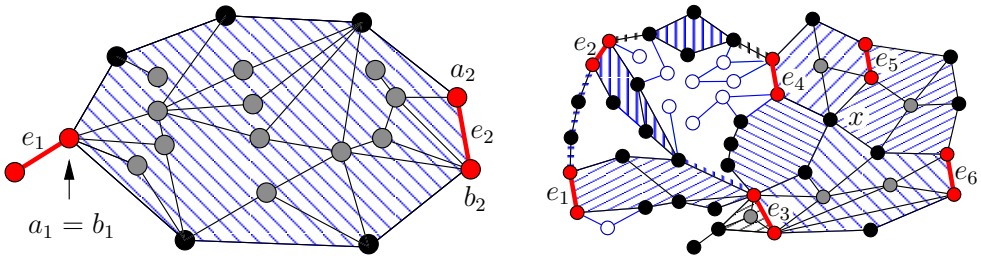


Figure 8.2: (a) Example of region $R(e_1, e_2)$ between two edges $e_1, e_2 \in M$. Note that e_1 is not part of R, but only its endpoint $a_1 = b_1$. The black vertices are the boundary vertices, and the gray vertices in the hatched area are the vertices strictly inside of R. (b) An example of an M-region decomposition: white vertices lie outside of regions and each region is hatched with a different pattern.

Definition 8.2. *Let G be a plane graph and let M be a maximum induced matching of G. An* M-region decomposition *of $G = (V, E)$ is a set $\mathcal{R}$ of regions such that no vertex in V lies strictly inside more than one region from $\mathcal{R}$. For an M-region decomposition $\mathcal{R}$, we define $V(\mathcal{R}) := \bigcup_{R \in \mathcal{R}} V(R)$. An M-region decomposition $\mathcal{R}$ is* maximal *if there is no $R \notin \mathcal{R}$ such that $\mathcal{R} \cup \{R\}$ is an M-region decomposition with $V(\mathcal{R}) \subsetneq V(\mathcal{R}) \cup V(R)$.*

For an example of an M-region decomposition, see Figure 8.2b.

Lemma 8.4 ([MS08a])**.** *Given a plane reduced graph $G = (V, E)$ and a maximum induced matching M of G, there exists an algorithm that constructs a maximal M-region decomposition with $O(|M|)$ regions.*

Bounding the Size of a Region

To upper-bound the size of a region R we make use of the fact that any vertex strictly inside R has distance at most two from some vertex in the boundary δR. For this reason, the vertices strictly inside R can be arranged in two layers. The first layer consists of the neighbors of boundary vertices, and the second of all the remaining vertices, that is, all vertices at distance at least two from every boundary vertex. The proof strategy is to show that if any of these layers contains too many vertices, then there exists an induced matching M' with $|M'| > |M|$. An important structure for our proof are areas enclosed by cycles of length four, called *diamonds*.

Definition 8.3. *Let u and v be two vertices in a plane graph. A* diamond $D(u, v)$[3] *is a closed area of the plane with two length-2 paths between u and v as boundary.*

[3] In standard graph theory, a diamond denotes a cycle of length four with exactly one chord. We abuse this term here. Note that diamonds also play an important role in proving linear problem kernels on planar graphs for other problems [AFN04, GN07b].

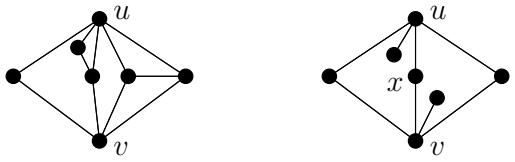


Figure 8.3: A diamond (left) and an empty diamond (right) in a reduced plane graph.

A diamond $D(u, v)$ is called empty *if every edge e in the diamond is incident to either u or v.*

Figure 8.3 shows an empty and a non-empty diamond. In a reduced plane graph, empty diamonds have restricted size. We are especially interested in the maximum number of vertices strictly inside an empty diamond $D(u, v)$ that have both u and v as neighbors.

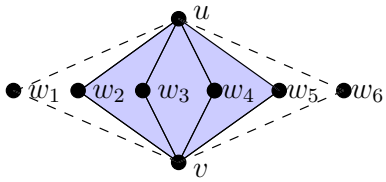
Lemma 8.5. *Let $D(u, v)$ be an empty diamond in a reduced plane graph. Then, there exists at most one vertex strictly inside $D(u, v)$ that has both u and v as neighbors.*

Proof. Suppose that there are at least two vertices x and y strictly inside $D(u, v)$, where both have u and v as neighbors. Since D is empty, x and y can have no other neighbors than u and v. Thus, there are two vertices of degree two with the same neighbors, a contradiction to the fact that G is reduced with respect to Reduction Rule 8.3. □

Lemma 8.5 shows that if there are more than three edge-disjoint length-two paths between two vertices u, v, then there must be an edge e in an area enclosed by two of these paths such that e is neither incident to u nor v. This fact is used in the following lemma to show that the number of length-two paths between two vertices of a reduced plane graph is bounded.

Lemma 8.6. *Let u and v be two vertices of a reduced plane graph G such that there exist two distinct length-two paths (u, x, v) and (u, y, v) between u and v enclosing an area A of the plane. Let M be a maximum induced matching of G. If neither x nor y is endpoint of an edge in M and no vertex strictly inside A is contained in $V(M)$, then there are at most* 15 *edge-disjoint length-two paths between u and v.*

Proof. The idea of the proof is to show that if there are more than 15 length-two paths between u and v, then we can exhibit an induced matching M' with $|M'| > |M|$, which would then contradict the optimality of M. First, we consider the case when neither u nor v is contained in $V(M)$. Suppose for the purpose of contradiction that there are six common neighbors $w_1, \ldots, w_6$ of u and v that lie inside A (that is, strictly inside and on the enclosing paths). Without loss of generality, suppose that these vertices are embedded as in the following figure:



Consider the diamond D with the boundary induced by the vertices u, v, w_2, and w_5. Since w_3 and w_4 are strictly inside D and are incident to both u and v, by Lemma 8.5, we know that D is not empty. That is, there exists an edge e in D which is not incident to u or v. Clearly e is incident to neither w_1 nor w_6 and the endpoints of e are at distance at least two from every vertex in $V(M)$. Therefore, we can add e to M and obtain a larger induced matching, which contradicts the optimality of M.

Next, consider the case when u and/or v are endpoints of edges in M. Using the same idea as above, it is easy to see that if there exist 16 length-two paths between u and v, then there are at least three non-empty diamonds (using $(u, w_1, v), (u, w_6, v)$ and (u, w_{11}, v) as "isolating paths") whose boundaries share only u and v. We can then replace the at most two edges in M incident with u and v by three edges, one from each nonempty diamond, and obtain a larger induced matching. □

Lemma 8.6 is needed to upper-bound the number of vertices inside and outside of regions that are connected to at least two boundary vertices. The next two lemmas are needed to upper-bound the number of vertices that are connected to exactly one boundary vertex. First, Lemma 8.7 upper-bounds the number of such vertices under the condition that they are contained in an area which is enclosed by a short cycle. Lemma 8.7 is then used in Lemma 8.8 to upper-bound the total number of such vertices for a given boundary vertex.

Lemma 8.7. *Let u be a vertex in a reduced plane graph G and let $v, w \in N(u)$ be two distinct vertices whose distance is at most three in $G - u$. Let P denote a shortest path between v and w in $G - u$ and let A denote the area of the plane enclosed by P and the path (v, u, w). If there are at least nine neighbors of u strictly inside A, then there is at least one edge strictly inside A.*

Proof. Let u contain nine neighbors $\{z_1, \ldots, z_9\}$ strictly inside A and assume that there is no edge strictly inside A. By Reduction Rule 8.2, at most one of the z_i's can have degree one. Without loss of generality, assume that z_9 has degree one. By Reduction Rule 8.3, no two degree-two vertices have the same neighborhood. Observe that the neighbors of the z_i's must be vertices on P due to planarity, as otherwise there would be an edge strictly inside of A, a contradiction to our assumption. First, consider the case when there exists a vertex among the z_i's of degree at least four. Suppose z_j, $1 \leq j \leq 8$, has at least three neighbors among the vertices in P. Because the graph is planar, there exists a vertex $x \in P$ such that no z_i, $i \neq j$, is adjacent to x. The remaining vertices have degree two or three

and each is adjacent to some vertex $y \neq x$ in P. Moreover, there can be at most one vertex of degree three. Since $|V(P)| \leq 4$, it is easy to see that there are at least two degree-two vertices with the same neighbors, a contradiction. Therefore, assume that $\deg(z_i) \leq 3$ for all i. Again by planarity, there are at most three vertices in $\{z_1, \ldots, z_8\}$ of degree three. The remaining at least five vertices must be of degree two and each is adjacent to a vertex in P. Since $|V(P)| \leq 4$, this implies that there are two degree-two vertices with the same neighborhood, a contradiction. This shows that if there exist nine neighbors of u in A, then there exists an edge strictly inside A. □

Lemma 8.8. *Let G be a reduced plane graph, let M be a maximum induced matching of G, let $e_1, e_2 \in M$ be edges that form a region $R(e_1, e_2)$, and let u be a boundary vertex of R. Then, u has at most* 40 *neighbors strictly inside R that are not adjacent to any other boundary vertex.*

Proof. We assume that there are 41 neighbors of u strictly inside R that are not adjacent to any other boundary vertex and show that then we can find an induced matching M' with $|M'| > |M|$, contradicting the maximum cardinality of M.

Suppose that the neighbors $v_1, \ldots, v_{41}$ are embedded around u in a clockwise fashion. By Reduction Rule 8.2, u can have at most one neighbor of degree one. Without loss of generality assume that $\deg(v_2) = 1$. Consider the vertices v_1, v_{11}, and v_{21}. If the pairwise distance of these vertices in $G-u$ is at least four, then any three edges e_a, e_b, e_c in $G-u$ incident to v_1, v_{11}, and v_{21}, respectively, are pairwise non-adjacent. Since they lie strictly inside $R(e_1, e_2)$ (u is the only neighbor on the boundary), we can set $M' := (M \setminus \{e_1, e_2\}) \cup \{e_a, e_b, e_c\}$. Similarly, if v_{21}, v_{31}, and v_{41} have a pairwise distance of at least four, then we can construct an induced matching of cardinality greater than $|M|$.

It remains to show the case that at least two vertices from $\{v_1, v_{11}, v_{21}\}$ have distance at most three and at least two vertices from $\{v_{21}, v_{31}, v_{41}\}$ have distance at most three. Let $\{w_1, w_1'\} \subseteq \{v_1, v_{11}, v_{21}\}$ and $\{w_2, w_2'\} \subseteq \{v_{21}, v_{31}, v_{41}\}$ be these vertices. Let P_1 and P_2 denote, respectively, the shortest paths from w_1 to w_1' and from w_2 to w_2' in $G-u$. Note that P_1 and P_2 are strictly inside R. Let A_1 be the area enclosed by P_1 and the path (w_1, u, w_1') and let A_2 be the area enclosed by P_2 and the path (w_2, u, w_2'). Note that P_1 and P_2 can be chosen so that the subsets of the plane strictly inside A_1 and A_2 do not intersect. By Lemma 8.7, there exist edges e_1, e_2 such that e_1 is strictly inside A_1 and e_2 is strictly inside A_2. If there exists an edge $e \in M$ incident to u, then $(M - e) \cup \{e_1, e_2\}$ is an induced matching with size strictly larger than that of M, a contradiction. If no edge of M is incident to u, then $M \cup \{e_1, e_2\}$ is again an induced matching of larger size. □

With Lemma 8.6 and Lemma 8.8, we now have the building blocks to prove the number of vertices inside a region. Recall that the vertices in a region can be arranged in two layers, one with all vertices with distance one to a boundary vertex, and one with distance two. First, the number of vertices in the first layer

is bounded. Then, to bound the number of vertices with at least two neighbors on the boundary, one can use Lemma 8.6 together with the Euler formula and the maximum number of vertices on the boundary (ten vertices). For the remaining vertices (that is, vertices with exactly one neighbor on the boundary), one can use Lemma 8.8. The number of vertices in the second layer is then bounded analogously, based on the bound for the first layer.

Lemma 8.9 ([MS08a])**.** *Each region $R(e_1, e_2)$ of an M-region decomposition of a reduced plane graph contains $O(1)$ vertices.*

Proposition 8.5. *Let G be a reduced plane graph and let M be a maximum induced matching of G. There exists an M-region decomposition such that the total number of vertices inside all regions is $O(|M|)$.*

Proof. Using Lemma 8.4, there exists a maximal M-region decomposition for G with at most $O(|M|)$ regions. By Lemma 8.9, each region has a constant number of vertices. Thus, there are $O(|M|)$ vertices inside regions. □

It remains to bound the number of vertices that lie outside regions of a maximal M-region decomposition. This proof uses similar ideas as the proof of Proposition 8.5; in particular, it also applies Lemma 8.6 and Lemma 8.7. Therefore, we omit the details here.

Proposition 8.6 ([MS08a])**.** *Given a maximal M-region decomposition with $O(|M|)$ regions, the number of vertices that lie outside of regions is $O(|M|)$.*

By Propositions 8.5 and 8.6, given a reduced plane graph G and a maximum induced matching M of G, there exists an M-region decomposition with $O(|M|)$ regions such that the number of vertices inside and outside of regions is $O(|M|)$. Therefore, since $|M| \leq k$, this shows the $O(k)$ upper bound on the number of vertices as claimed in Theorem 8.1, that is, Induced Matching admits a linear problem kernel on planar graphs. Note that the constant hidden in the O-Notation is rather big; we estimate it to be at least above 1000. The $40k$-size problem kernel by Kanj et al. [KPXS09] is clearly much smaller.

8.2.3 Graphs of Bounded Treewidth

Zito [Zit99] developed a linear-time dynamic programming algorithm to solve Induced Matching on trees. We extend his work and obtain a linear-time algorithm on graphs of bounded treewidth (see Section 2.3.4). Note that compared to Zito's work our dynamic programming approach uses a different encoding to store the partial solutions in the updating process. Before showing our algorithm, we briefly outline how one can verify that such a linear-time algorithm for graphs of bounded treewidth actually does exist. However, unlike our algorithm, this method does not derive an algorithm with good running times.

Proposition 8.7. *Given an undirected graph with a tree decomposition of width at most $\omega \geq 1$, then* Induced Matching *can be solved in linear time.*

Proof. We apply a result by Courcelle [Cou90]. It states that all graph properties definable in monadic second-order logic (MSO) can be decided in linear time on graphs of bounded treewidth. There are extensions of MSO allowing us to deal with optimization problems. We give an MSO formulation of (the optimization version of) Induced Matching:

$$\max E' : \forall e_1 \forall e_2 \big(E' e_1 E' e_2 \neg \big[\exists x \exists y V x \wedge V y \wedge I x e_1 \wedge I y e_2 \wedge \\ ((x = y) \vee \exists e' (E e' \wedge I x e' \wedge I y e')) \big] \big)$$

In the above formula, V and E are unary relation symbols which denote the vertex and edge sets of the graph; I is a binary relation symbol that denotes whether a vertex is incident to an edge and E' denotes an induced matching. The formula expresses that for each edge pair in the induced matching (e_1, e_2) there must not exist two vertices $x \in e_1$ and $y \in e_1$ such that $x = y$ or x and y are adjacent (in other words, the edges e_1 and e_1 must not overlap nor be connected by another edge). □

Courcelle's result is of purely theoretical interest as the hidden constants in the running time analysis are huge. For this reason, it is of independent interest to develop practical algorithms. It is relatively easy to see that a standard dynamic programming approach would result in a running time of $O(9^\omega \cdot n)$, where ω is the width of the given tree-decomposition. With an improved dynamic programming algorithm, we obtain a running time of $O(4^\omega \cdot n)$. Our approach also uses some ideas that were applied for an improved dynamic programming algorithm for Dominating Set [AN02, ABF+02]. However, the concept of monotonicity introduced by Alber and Niedermeier [AN02] for Dominating Set is not needed for Induced Matching, as the necessary condition for an improved analysis of the dynamic programming update process is fulfilled without the monotonicity concept. Here we describe only the basic definitions and those parts of the algorithm that are important in showing the improved running time.

The remainder of this section is dedicated to the proof of the following theorem.

Theorem 8.2. *Let $G = (V, E)$ be a graph with a given nice tree decomposition $(\{X_i \mid i \in I\}, T)$ of width ω. Then the size of a maximum induced matching of G can be computed in $O(4^\omega \cdot n)$ time, where $n := |I|$.*

Proof. For each bag X_i we consider all possible ways of obtaining an induced matching in the subgraph induced by X_i and all bags below X_i. To do this, we create a table $A_i, i \in I$ for each bag X_i, which stores this information. These tables are updated in a bottom-up process starting at the leaves of the decomposition tree. In the following, we say that a vertex v is *contained* in an induced

matching M if v is an endpoint of an edge in M. If v is contained in M, then its *partner* in M is the vertex u such that $\{u, v\} \in M$. We use different colors to represent the possible states of a vertex in a bag:

white(0): A vertex labeled 0 is not contained in M.

black(1): A vertex labeled 1 is contained in M and its partner in M has already been discovered in the current stage of the algorithm.

gray(2): A vertex labeled 2 is contained in M but its partner in M has not been discovered in the current stage of the algorithm.

For each bag $X_i = \{x_{i_1}, \ldots, x_{i_{n_i}}\}$, $|X_i| = n_i$, we construct a table A_i consisting of 3^{n_i} rows and $n_i + 1$ columns. Each row represents a coloring $c : X_i \to \{0, 1, 2\}$ of the graph $G[X_i]$; the entry $m_i(c)$ in the $n_i + 1$st column represents the number of vertices in an induced matching in the graph visited up to the current stage of the algorithm under the assumption that the vertices in the bag X_i are assigned colors as specified by c. If no induced matching is possible with the corresponding coloring, then the entry $m_i(c)$ stores the value $-\infty$. For a coloring $c = (c_1, \ldots, c_m) \in \{0, 1, 2\}^m$ and a color $d \in \{0, 1, 2\}$ we define $\#_d(c) := |\{1 \leq t \leq m : c_t = d\}|$.

Given a bag X_i and a coloring c of the vertices in X_i, we say that c is *valid* if the subgraph induced by the vertices labeled 1 and 2 has the following structure: vertices labeled 2 have degree 0 and those labeled 1 have degree ≤ 1. For valid colorings we store the value m_i as described above; for all other colorings we set m_i to $-\infty$ to mark it as invalid. A coloring is *strictly valid* if it is valid and, in addition, vertices labeled 1 induce isolated edges. We next describe the dynamic programming process. Recall that we assume that we work with a nice tree decomposition.

Leaf Nodes. For a leaf node X_i compute the table A_i as

$$m_i(c) := \begin{cases} \#_1(c) + \#_2(c), & \text{if } c \text{ is strictly valid,} \\ -\infty, & \text{otherwise.} \end{cases}$$

In the initialization step, the assignment of colors needs to be justified locally and therefore we require that the colorings are *strictly* valid. Checking for validity takes $O(n_i^2)$ time; therefore, this step can be carried out in $O(3^{n_i} \cdot n_i^2)$ time.

Introduce Nodes. Let $X_i = \{x_{i_1}, \ldots, x_{i_{n_j}}, x\}$ be an introduce node with child node $X_j = \{x_{i_1}, \ldots, x_{i_{n_j}}\}$. Compute the table A_i as follows. For a coloring $c : X_i \to \{0, 1, 2\}$ and an index $1 \leq p \leq |X_i|$, define $\text{gray}_p(c)$ to be a coloring derived from c by re-coloring the vertex with index p with color 2. Let $N_j(x)$ be the set of neighbors of vertex x in X_j, that is, $N_j(x) := N(x) \cap X_j$.

Then the mapping m_i in A_i is computed as follows (recall that m_i represents the number of *vertices* in an induced matching in the graph visited up to the current stage of the algorithm). For a coloring $c = (c_1, \ldots, c_{n_j})$ set

$$m_i(c \times \{0\}) := m_j(c). \tag{8.1}$$

$$m_i(c \times \{1\}) := \begin{cases} m_j(\text{gray}_p(c)) + 1, & \text{if there is a vertex } x_{j_p} \in N_j(x) \\ & \text{with } c_p = 1, \text{ and for all} \\ & x_{j_q} \in N_j(x) \text{ with } q \neq p : c_q = 0. \\ -\infty, & \text{otherwise.} \end{cases} \tag{8.2}$$

$$m_i(c \times \{2\}) := \begin{cases} m_j(c) + 1, & \text{if } c_p = 0 \text{ for all } x_{j_p} \in N_j(x). \\ -\infty, & \text{otherwise.} \end{cases} \tag{8.3}$$

Assignment 8.1 is correct, since the coloring $c \times \{0\}$ is valid for X_i if and only if c is valid for X_j. The value of m_i is the same for both colorings. If the newly introduced vertex x has color 1 (Assignment 8.2), then—since $c \times \{1\}$ must be valid—there must be a neighbor y with color 1 within the bag X_i; all the other neighbors of x in X_i must have color 0. This is insured by the assignment condition. To see the correctness of the computed value $m_i(c \times \{1\})$, note that y must have color 2 in bag X_j, since the partner of y was not yet known in the stage when the algorithm was processing bag X_j, and we increase the number of solution vertices by one since the newly introduced vertex has color 1. The condition of Assignment 8.3 simply verifies the validity of the coloring $c \times \{2\}$, and we increase the number of solution vertices by one since the newly introduced vertex has color 2.

For each row of table A_i, we have to look at the neighborhood of vertex x within the bag X_i to check whether the corresponding coloring is valid. Therefore, this step can be carried out in $O(3^{n_i} \cdot n_i)$ time.

Forget Nodes. Let $X_i = \{x_{i_1}, \ldots, x_{i_{n_i}}\}$ be a forget node with child node $X_j = \{x_{i_1}, \ldots, x_{i_{n_i}}, x\}$. Compute the table A_i as follows. For each coloring $c \in \{0,1,2\}^{n_i}$ set

$$m_i(c) := \max_{d \in \{0,1\}} \{m_j(c \times \{d\})\}.$$

The maximum is taken over colors 0 and 1 only, as a coloring $c \times \{2\}$ cannot be extended to a maximum induced matching. To see this, note that such a coloring assigns vertex x color 2 and since x is forgotten, by the consistency property of tree decompositions, it does not appear in any of the bags that the algorithm sees in the future.

Clearly, this evaluation can be done in $O(3^{n_i} \cdot n_i)$ time. The crucial part are the join nodes.

Join Nodes. For a join node X_i with child nodes X_j and X_k compute the table A_i as follows. We say that two colorings $c' = (c'_1, \ldots, c'_{n_i}) \in \{0,1,2\}^{n_i}$

and $c'' = (c''_1, \ldots, c''_{n_i}) \in \{0,1,2\}^{n_i}$ are *correct* for a coloring $c = (c_1, \ldots, c_{n_i})$ if the following conditions hold for every $p \in \{1, \ldots, n_i\}$:

1. if $c_p = 0$ then $c'_p = 0$ and $c''_p = 0$,
2. if $c_p = 1$ then
   (a) if x_{i_p} has a neighbor $x_{i_q} \in X_i$ with $c_q = 1$ then $c'_p = c''_p = 1$,
   (b) else either $c'_p = 1$ and $c''_p = 2$, or $c'_p = 2$ and $c''_p = 1$, and
3. if $c_p = 2$ then $c'_p = 2$ and $c''_p = 2$.

Finally, the mapping m_i of X_i is evaluated as follows. For each coloring $c \in \{0,1,2\}^{n_i}$ set

$$m_i(c) := \max\{m_j(c') + m_k(c'') - \#_1(c) - \#_2(c) \mid c' \text{ and } c'' \text{ are correct for } c\}.$$

In other words, we determine the value of $m_i(c)$ by looking up the corresponding coloring in m_j and in m_k (corresponding to the left and right subtree, respectively), add the corresponding values and subtract the number of vertices colored 1 or 2 by c, since they would be counted twice otherwise.

Clearly, if the coloring c assigns color 0 to a vertex $x \in X_i$, then so must colorings c' and c''. The same holds if c assigns color 2 to a vertex. However, if c assigns color 1 to a vertex x, then this coloring can be justified in two ways. The first case is when x has a neighbor $y \in X_i$ that is also colored 1. Then both colorings c' and c'' obviously assign 1 to x (and 1 to y). The second case is when all neighbors of x in X_i are assigned color 0. Then the assignment $c(x) = 1$ must be justified by another vertex in the solution which is in a bag which has already been processed in a previous stage of the algorithm. This vertex is located either in the left subtree or in the right subtree (corresponding to m_j or m_k, respectively), but not both. Therefore, the color of x can only be justified by assigning color 1 to x by c' and color 2 to x by c'', or vice versa.

Note that for a given coloring $c \in \{0,1,2\}^{n_i}$, with $a := \#_1(c)$, there are at most 2^a possible pairs of correct colorings for c. There are $2^{n_i-a}\binom{n_i}{a}$ possible colorings c with a vertices colored 1, thus

$$\begin{aligned}|\{(c', c'') \mid c \in \{0,1,2\}^{n_i}, c' \text{ and } c'' \text{ are correct for } c\}| &\leq \sum_{a=0}^{n_i} 2^{n_i-a}\binom{n_i}{a} \cdot 2^a \\ &= 4^{n_i}.\end{aligned}$$

Since we have to check the neighbors of x within X_i for each pair of correct colorings, the total running time for this step is $O(4^{n_i} \cdot n_i)$. In total, we get a running time of $O(4^\omega \cdot |I|)$ for the whole dynamic programming process. □

8.3 Outlook

We determined the parameterized complexity of Induced Matching for planar graphs, graphs of bounded degree, bipartite graphs, graphs with girth at least six, line graphs, and graphs of bounded treewidth. The data reduction rules for the planar case are very simple and the kernelization can be done in linear time. It seems plausible that the technique used for the kernel on planar graphs (especially the layer-wise analysis of the size of a region, which is not available in the proof of the $40k$-size problem kernel by Kanj et al. [KPXS09]) can be generalized to achieve kernelization results for Distance-d Matching.

Another possible future research topic is graph packing with distance properties; the basic task is packing graphs as in Chapter 7, but with the additional constraint that the packed graphs must have pairwise distance d for some fixed d. This problem generalizes Distance-d Matching. It has not been investigated as intensively as Induced Matching, but few studies already exist for Triangle Packing with distance two [CH06, Cam09]. Quite surprisingly, it turns out that this packing problem with distance constraint is polynomial-time solvable in some graph classes, while the packing problem without distance constraint is NP-complete [CH06]. It would be interesting to see whether that sort of results (that is, that the distance constraint, unlike for Induced Matching, makes the problem easier) is also possible in the context of parameterized complexity; maybe combining techniques for packing problems (see Chapter 7) and ideas from this chapter could be a good starting point.

Chapter 9

Implementation and Experiments for Finding Maximum s-Plexes

In this chapter, we propose new practical algorithms to find degree-relaxed variants of cliques called s-plexes. An s-plex denotes a vertex subset in a graph inducing a subgraph where every vertex has edges to all but at most $s-1$ other vertices in the s-plex. Cliques are 1-plexes. In analogy to the special case of finding maximum-cardinality cliques, finding maximum-cardinality s-plexes is NP-hard [BBH09]. Complementing previous work, we develop combinatorial, exact algorithms, which are strongly based on the close connection between finding maximum-cardinality s-plexes and the Bounded-Degree-d Vertex Deletion problem (cf. Chapter 4). The experiments with our freely available implementation indicate the competitiveness of our approach, for many real-world graphs outperforming the previously used methods.

First, we introduce the problem of finding maximum-cardinality s-plexes, describe its relation to Bounded-Degree-d Vertex Deletion, and list some known results (Section 9.1). Then, we describe our algorithmic approach in more detail (Section 9.2) and report about the implementation and algorithmic tricks (Section 9.3). After that, we report about the experimental results (Section 9.4). Finally, we give further remarks (Section 9.5) and an outlook (Section 9.6).

9.1 Introduction and Known Results

Finding maximum-cardinality cliques in graphs for a long time has been a major challenge for algorithmic graph theory and corresponding algorithm engineering efforts (cf. DIMACS challenge [DIM95]). The corresponding Maximum Clique problem is NP-hard and neither effective approximation nor parameterized approaches exist that allow for efficient algorithms with provable performance bounds. Hence, the use of heuristic approaches always has been an important tool for practical solutions of Maximum Clique. The concept of cliques, however,

has been criticized for its overly restrictive nature asking for *complete* subgraphs, since in many applications one searches for dense subgraphs that do not have to meet the "perfect" notion of cliques and could be missing some edges. A more relaxed concept of a dense subgraph has been introduced by Seidman and Foster [SF78] with the notion of s-plexes. A 1-plex is the same as a clique.

Definition 9.1. *For $s \geq 2$, an* s-plex *is a graph $G = (V, E)$ in which each vertex has degree at least $|V| - s$.*

In other words, in an s-plex each vertex is adjacent to all but at most $s - 1$ other vertices. Unfortunately, finding maximum-cardinality s-plexes turns out to be computationally basically as hard as clique detection is [BBH09, KHMN09]. Thus, recently the development of practical (heuristic) algorithms for s-plex detection has received quite some interest [BBH09, MH09, WP07] although being still in its infancy compared to the algorithm engineering undertaken for clique finding. In this work, we contribute novel tools for the efficient detection of maximum-cardinality s-plexes. Other than previous work [BBH09, MH09, WP07] (where [WP07] deals with s-plex *enumeration*), our algorithms draw on methods from parameterized algorithmics.

The Maximum s-Plex problem for an integer $s \geq 1$ is defined as follows.

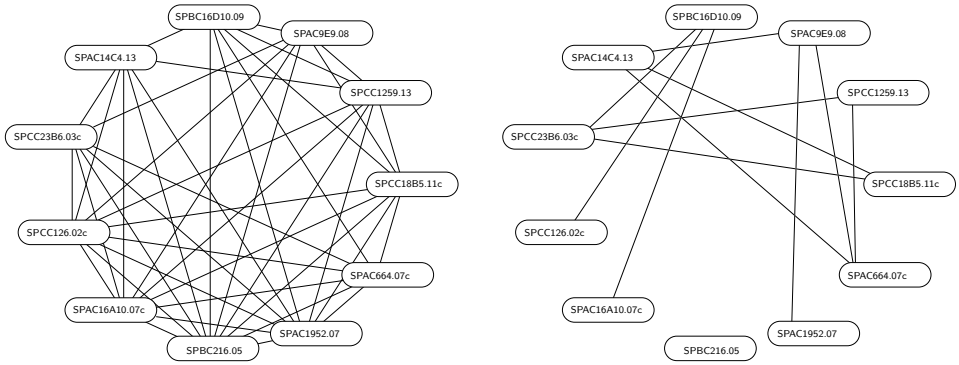
> Maximum s-Plex
> **Input:** An undirected graph $G = (V, E)$ and an integer $k \geq 0$.
> **Question:** Does there exist a vertex subset $S \subseteq V$ of size at least k such that $G[S]$ is an s-plex?

Clearly, in our experiments we actually choose to maximize the value of k. Recent work on clique finding has exploited the close connection (indeed, duality) between Maximum Clique and the Minimum Vertex Cover problem [ACF+04, AFLS07, CLS+05]. We follow the same spirit here and make use of the duality between Maximum s-Plex and the Bounded-Degree-d Vertex Deletion problem. Bounded-Degree-d Vertex Deletion is the problem considered in Chapter 4; we repeat its definition.

> Bounded-Degree-d Vertex Deletion
> **Input:** An undirected graph $G = (V, E)$ and an integer $k \geq 0$.
> **Question:** Does there exist a vertex subset $S \subseteq V$ of size at most k such that $G - S$ has maximum degree d?

Clearly, we are interested in minimizing the value k. The duality between Maximum s-Plex and Bounded-Degree-d Vertex Deletion can be expressed as follows. Recall that a bdd-d-set is a solution set for Bounded-Degree-d Vertex Deletion.

Lemma 9.1. *A graph $G = (V, E)$ contains an s-plex of size k if and only if the complement graph $\bar{G} = (V, \bar{E})$ contains a bdd-d-set of size $|V| - k$ with $d := s - 1$.*



(a) A maximum 4-plex in a protein-protein interaction graph.

(b) Corresponding complement.

Figure 9.1: (a) Maximum-cardinality 4-plex in a fission yeast protein-protein interaction graph and (b) the corresponding complement, which has bounded degree three. The vertices of the protein-protein interaction graph correspond to proteins, and an edge $\{u, v\}$ indicates an experimentally determined interaction between the proteins corresponding to u and v. The entire graph has 1053 vertices and 2884 edges and was generated using data from the BioGRID database (`http://www.thebiogrid.org`).

Proof. ($\Rightarrow$) Let S be an s-plex of size k in G. Every vertex in $G[S]$ has degree at least $|S| - s$, that is, each vertex in $G[S]$ is adjacent to all but at most $s - 1$ other vertices in $G[S]$. Therefore, each vertex in $\bar{G}[S]$ has degree at most $s - 1$. Hence, $V \setminus S$ is a size-$(|V| - k)$ bdd-d-set for $\bar{G}$, where $d = s - 1$.

($\Leftarrow$) Let S' be a bdd-d-set of size $|V| - k$ for $\bar{G}$. Each vertex in $\bar{G} - S'$ has degree at most $d = s - 1$ and, therefore, $G - S'$ is an s-plex of size k. □

See Figure 9.1 for an s-plex in a real-world graph from computational biology (in which dense subgraphs correspond to functional modules, see, e.g., [BBT05]) together with the corresponding complement of bounded degree $s - 1$. We exploit this close connection by making use of fixed-parameter tractability results for Bounded-Degree-d Vertex Deletion (Chapter 4). By the proof of Lemma 9.1, a minimum-cardinality bdd-d-set can then directly be transformed into a corresponding maximum-cardinality s-plex.

Known Results. The s-plex concept was introduced by Seidman and Foster [SF78] in the context of social network analysis, accompanied by some structural results. Maximum s-Plex is NP-complete, which follows directly from the duality between Maximum s-Plex and Bounded-Degree-d Vertex Deletion (Lemma 9.1); Bounded-Degree-d Vertex Deletion is NP-complete due to a general framework by Lewis and Yannakakis [LY80] (cf. Chapter 4). Balasundaram et al. [BBH09] also gave a direct NP-completeness proof by reduction from Maximum Clique. Concerning parameterized complexity, it is known that

Maximum s-Plex is W[1]-hard with respect to the parameter k [KHMN09]. The s-Plex Editing problem, that is, the problem of obtaining an s-plex graph by at most k edge deletions and additions, admits a problem kernel of $(4s^2-2)\cdot k + 4(s-1)^2$ vertices for $s \geq 2$, which can be constructed in $O(n^4)$ time [GKNU09]. Moreover, s-Plex Editing can be solved in $O((2s+\lfloor\sqrt{s}\rfloor)^k+n^4)$ time [GKNU09].

Balasundaram et al. [BBH09] presented a 0/1 integer linear program for Maximum s-Plex, generalizing a known formulation for the special case Maximum Clique. In addition, they carried out a polyhedral study of the problem and discussed a branch-and-cut implementation as the basis of computational tests. McClosky and Hicks [MH09] described combinatorial algorithms for Maximum s-Plex, both of heuristic (without provable guarantees on the solution quality) and exact nature. Their heuristic algorithms are based on certain upper and lower bounds for vertex coloring and their exact algorithms are based on adapting known algorithms for Maximum Clique. Both papers implement their algorithms and do computational experiments with real-world and artificial graph instances. The two papers [BBH09, MH09] are the main comparison points with our implementation presented in this chapter. Wu and Pei [WP07] gave an algorithm to *enumerate* all maximal s-plexes in a graph, also accompanied by experimental studies.

Related Clique Relaxations. Other relaxations of the clique concept are for instance s-cores [SF78], s-clubs [BBT05], s-cliques [BBT05], paracliques [CLS+05], pseudo cliques [II09], and defective cliques [YPTG06]. Komusiewicz [Kom07] gave a short overview on most of these concepts.

9.2 Algorithmic Approach

Our approach to compute maximum-cardinality s-plexes makes use of the duality between Maximum s-Plex and Bounded-Degree-d Vertex Deletion (Lemma 9.1). Therefore, the first step is to compute the complement of the input graph[1]. Then, the second step is to solve Bounded-Degree-d Vertex Deletion on that complement graph for $d := s-1$. Finally, the minimum bdd-d-set is translated back into a maximum s-plex in the input graph.

In the following, we assume that the graph G is the input to the Bounded-Degree-d Vertex Deletion problem. We suppose that the parameter k denoting the number of allowed vertex deletions is given (see Section 9.3 for details of how k is obtained). Our main algorithm to solve Bounded-Degree-d Vertex Deletion uses a bounded search tree and polynomial-time data reduction rules interleaving with the search tree. In general, the branching strategy of the search tree algorithm chooses a vertex v of degree at least $d+1$, and then branches

[1]Note that we avoid a potentially quadratic blow-up of the edge number (in the complement graph) by only simulating the complement graph rather than constructing it.

into the subcases of deleting v and every possibility of deleting all but d neighbors of v. We refer to this by saying that the strategy "branches on v and $N(v)$". A theoretical analysis of such a bounded search tree is given in Section 4.4.1. In practice, it is favorable to delete many vertices in each branching step, that is, v should be a vertex of high degree.

In the next section, we describe the data reduction rules that are applied in each search tree node (Section 9.2.1) and some heuristic improvements for the search tree approach (Section 9.2.2).

9.2.1 Data Reduction Rules

The best problem kernel for Bounded-Degree-d Vertex Deletion in terms of kernel size is the "almost linear" problem kernel of $O(k^{1+\epsilon})$ vertices (for any constant $\epsilon > 0$) that follows directly from the local optimization theorem (Theorem 4.2 in Section 4.3). We deviate from the local optimization algorithm in Section 4.3 and give a heuristic data reduction rule based on Theorem 4.2, because preliminary experiments showed that a direct implementation of the algorithm in Section 4.3 has less chance to successfully reduce the graphs we considered (basically due to the constant in the kernel size and the fact that the parameter k is not very small in these instances). Before that, we describe some other simple reduction rules.

The following data reduction rules find vertices that can be safely assumed to be in an optimal bdd-d-set of size at most k for the graph G. The algorithm maintains a set S which is used to store the vertices that are already deleted by the reduction rules. This is necessary to be able to return the complete bdd-d-set at the end.

High-Degree Rule. If a vertex v of degree more than $d + k$ is found, then delete v from G, decrease the parameter k by one, and add v to S. This is exactly Reduction Rule 4.1 in Section 4.2, which is used to obtain an $O(k^2)$-vertex kernel for Bounded-Degree-d Vertex Deletion. For the correctness of this rule, see Section 4.2. We mention in passing that the "low-degree rule", that is, to delete a vertex v if each vertex in $N[v]$ has degree at most d (Reduction Rule 4.2 in Section 4.2), is not used, as preliminary experiments showed that this rule has almost no effect in practice or may even slow down the algorithm.

Degree-One Rule. This data reduction rule deletes vertices that have many degree-one neighbors.

Reduction Rule 9.1. *If a vertex $v \in V$ has at least $d+1$ degree-one neighbors, then delete v from G, add v to S, and decrease k by one.*

Lemma 9.2. *Reduction Rule 9.1 is correct and can be exhaustively applied in $O(n + m)$ time.*

Proof. First, concerning the correctness, observe that at least v or one of its degree-one neighbors has to be in an optimal solution. If a degree-one neighbor w of v is in an optimal bdd-d-set S, but $v \notin S$, then one can simply remove w from S and add v to it, obtaining a bdd-d-set of the same size. Therefore, it is always safe to assume that an optimal solution contains v.

It remains to show the running time. First, initialize an empty list for each vertex v in the graph, which is used to store the degree-one neighbors of v. Then, iterate over all vertices and collect all degree-one vertices in $O(n)$ time. For each degree-one vertex w with neighbor v, add w to the list of v. If in this process one encounters a vertex v whose list contains at least $d+1$ vertices, delete v from the graph. Some neighbors of v might obtain degree one by v's deletion; such degree-one vertices are also processed as described. Deleting a vertex v takes $O(|N(v)|)$ time; thus, the deletion of vertices takes $O(n+m)$ time in total. The entire process therefore takes $O(n+m)$ time. □

BDD-NT-Rule. This rule is a heuristic version of the $O(k^{1+\epsilon})$-vertex kernel (for any constant $\epsilon > 0$) that is given by Theorem 4.2 in Section 4.3. The core of the proof of Section 4.3 is the algorithm FindExtremal. The input for FindExtremal is a graph $G = (V, E)$ and a bdd-d-set X. For $d \leq 1$, if $|V \setminus X| > (d+1)^2 \cdot |X|$, then FindExtremal returns in $O(n^3 \cdot m)$ two vertex subsets $A' \subseteq X$ and $B' \subseteq V \setminus X$. The sets A' and B' have the property that for vertices in A' it can already be decided to put them into the solution set and vertices in B' can be ignored for finding a solution, and B' is not empty (thus, we have a guarantee that a reduction rule based on FindExtremal will successfully reduce the graph). Thus, assuming that X is given, one can simply test whether the condition $|V \setminus X| > (d+1)^2 \cdot |X|$ is fulfilled, and if so, apply FindExtremal. The resulting set A' can be safely assumed to be in a minimum-cardinality bdd-d-set of G, and we can add A' to the solution set S and delete A' from G, decreasing the parameter by $|A'|$. The vertices in B' do not have to be considered and can be deleted from the graph. The same principle also works for $d \geq 2$: if $|V \setminus X| = \omega(|X|^{1+\epsilon})$, then FindExtremal returns in $O(n^3 \cdot m)$ time two vertex subsets $A' \subseteq X$ and $B' \subseteq V \setminus X$ with the properties that for vertices in A' it can already be decided to put them into the solution set and vertices in B' can be ignored for finding a solution, and B' is not empty, and the same reduction as in the case $d \leq 1$ can be applied. However, for the graphs we consider in this chapter, the constant hidden in the O-notation of the kernel size bound turns out to be too big. The larger constant compared to the case $d \leq 1$ is due to the fact that FindExtremal it based on stars with $d+1+\lceil |X|^{\epsilon} \rceil$ leaves for $d \geq 2$ instead of $d+1$ leaves as in the case $d \leq 1$. However, without giving the details here, FindExtremal can compute two sets A' and B' such that A' can be safely assumed to be in a bdd-d-set of G and the vertices of B' can be ignored for finding a solution also if one uses stars with $d+1$ leaves for $d \geq 2$. The drawback of using $d+1$ stars for $d \geq 2$ is that then the returned set B'

Algorithm: BddReduction (G, X, k)
Input: A graph $G = (V, E)$, a bdd-d-set X for G, and an integer $k \geq 0$.
Output: A reduced instance (G, k), a modified bdd-d-set X for G, a set S of reduced vertices.

```
1  S ← ∅
2  repeat
3      while ∃v ∈ V : deg(v) > d + k                          ▷ High-degree rule
4          G ← G − v; X ← X \ {v}; S ← S ∪ {v}; k ← k − 1.
5      while ∃v ∈ V : v has at least d + 1 deg-1 neighbors    ▷ Degree-1 rule
6          G ← G − v; X ← X \ {v}; S ← S ∪ {v}; k ← k − 1.
7      if |N(X)| > (d + 1) · |X| then
8          call BDD-NT-rule to obtain vertex sets C and D
9          G ← G − (C ∪ D); X ← X \ C; S ← S ∪ C; k ← k − |C|
10 until none of the rules applies.
11 return G, k, X, S
```

Figure 9.2: Pseudo-code of the basic algorithm to exhaustively apply the data reduction rules.

might be empty (and thus we have no provably effective reduction rule anymore). By preliminary experiments, we found out that in practice it is very likely that FindExtremal terminates outputting two non-empty sets A' and B' with the above-mentioned properties if $|N(X)| > (d+1) \cdot |X|$, using stars with $d+1$ leaves for any constant d. This is the approach we are using in practice, although it is possible to construct instances where this adapted heuristic version would not reduce the graph.

For the interleaving with the search tree, we initially compute a $(d + 2)$-approximate solution X, and then start branching, always keeping X up-to-date, that is, if a vertex is deleted from the graph in the course of the branching process, then also delete it from X. Then, in each search tree node simply test whether $|N(X)| > (d + 1) \cdot |X|$, and, if so, then apply the modified FindExtremal as described above to compute A' and B' and use these sets to reduce the graph.

We apply the given data reduction rules exhaustively in each search tree node in the given order. The reason for this order is that the high-degree rule turns out to be the most effective in terms of number of deleted vertices and, at the same time, it can be implemented to run very efficiently. The other two rules are less effective in general, and the BDD-NT-rule is rather expensive in terms of running time, so it is the last rule that is considered.

Figure 9.2 presents the pseudocode of the implemented exhaustive data reduction based on the three stated data reduction rules. The code in lines 2–10 applies the high-degree rule (lines 3–4), the degree-one rule (lines 5-6), and the

BDD-NT-rule (lines 7–9), until none of the rules can be applied anymore. Note that the rules not only have to delete vertices from the graph G, but also from the bdd-d-set X in order to preserve the invariant that X is a bdd-d-set for G.

9.2.2 Heuristic Improvements

In the following, we describe some heuristics to speed up the search tree algorithm.

While branching, the search tree algorithm maintains a bdd-d-set X which is based on a $(d+2)$-approximate solution.[2] This vertex set X is an upper bound of the size of an minimum-cardinality bdd-d-set. When a vertex is deleted in the course of the branching, the bdd-d-set X is updated accordingly. First, we give a heuristic, called "guided branching", which tries to select vertices to branch on such that X becomes small very quickly in the course of the branching process.

After describing the "guided branching" heuristic, we give two heuristics used to computed lower bounds on the size of a minimum bdd-d-set.

Guided branching. As outline before, the guided branching heuristics aims to "guide" the search tree algorithm to select vertices to branch on such that a bdd-d-set X becomes small very quickly in the course of the branching process. There are two reasons why X should be small:

1. The set X is an upper bound on the size of a minimum-cardinality bdd-d-set and can be used to speed up the search, e.g., by setting $k := |X|$ if $|X| < k$ in some search tree node.

2. The BDD-NT-rule is based on X; the BDD-NT-rule can only be effective if X is small compared to $N(X)$ (we only apply the BDD-NT-rule if $|N(X)| > (d+1) \cdot |X|$, see also Figure 9.2).

That is why it is beneficial when the branching strategy tends to branch on vertices in X (thereby deleting more vertices in X) such that after few branching steps X gets small enough. However, in order to faster decrease the size of X, it can be useful to branch on v and only a subset of $N(v)$ (which, of course, only works if the subset contains more than d vertices). To this end, for a vertex v of maximum degree we branch on v and $N(v) \cap X$ if $|N(v) \cap X|/|N(v)| > 0.9$ (the value 0.9 has shown good results in preliminary experiments); otherwise, we simply branch on v and $N(v)$. The condition guarantees that we only branch on v and $N(v) \cap X$ if $N(v) \cap X$ contains at least 90% of the vertices in $N(v)$. The reason for this condition is to prevent that the algorithm branches on $N(v) \cap X$ if it is too small; in this case, the benefit of reducing only X does not outweigh the less effective branching when branching on v and a subset of its neighbors compared with branching on v and all its neighbors.

[2] This $(d+2)$-approximate solution can be found by greedily computing a maximal collection of vertex-disjoint copies of stars with $(d+1)$ leaves.

Edge-Count Test. The *edge-count test* checks whether the given Bounded-Degree-d Vertex Deletion instance is a no-instance. The test counts how many edges can be deleted from the graph $G = (V, E)$ by at most k vertex deletions based on the vertex degree distribution of the graph. If the number of such edges is too small, then the graph cannot be turned into a graph with maximum degree d by at most k vertex deletions. The number of edges m' that can be removed by k vertex deletions is computed as follows: sort the vertices of G by non-decreasing degree and sum up the degrees of the first k vertices in that order. Then, test whether

$$m - m' > \frac{dn}{2}.$$

If so, then (G, k) is a no-instance, since the minimum number of edges that remain in the graph after at most k vertex deletions is greater than the maximum number of edges that are allowed in an n-vertex graph of maximum degree d. Due to its simplicity, this test can be implemented to run very efficiently.

Lower Bound Heuristic. In order to derive a lower bound on the size of a bdd-d-set, we greedily compute a packing of vertex-disjoint stars with $d+1$ leaves. Since an optimal bdd-d-set has to contain at least one vertex of each star, the number of stars is a lower bound. This lower bound is used in the search tree algorithm to rule out branches that cannot yield an optimal solution.

9.2.3 The Search Tree Algorithm

In Figure 9.3, we give the pseudo-code of the basic search tree algorithm to compute a minimum bdd-d-set of size at most k for a graph including the data reduction rules and the heuristic improvements. In line 1 the algorithm applies the data reduction rules by calling algorithm BddReduction in Figure 9.2. In lines 2–5 we perform several tests whether the instance resulting by the application of the data reduction rules is a no-instance. In line 5 we test whether the instance has already bounded degree d. Then, in line 6 the algorithm selects a vertex to branch on. The branching is then performed in lines 9–11. To see the correctness of the branching, observe that the algorithm selects a vertex v and a subset $N' \subseteq N(v)$ of its neighbors, where $|N'| > d$. The algorithm branches into the case of deleting v from G and into all cases of deleting all but d vertices in N'. This generalizes the branching strategy we analyzed theoretically in Section 4.4.1, where the set N' has always size $d + 1$ and the algorithm branches into the case of deleting v and into $d + 1$ cases of deleting a vertex in N'. Then, in lines 12–14 the algorithm either returns that the input instance is a no-instance or returns the best solution that it has found.

Algorithm: BddSolve (G, X, k)
Input: A graph $G = (V, E)$, a bdd-d-set X for G, and an integer $k \geq 0$.
Output: A minimum-size bdd-d-set S for G with $|S| \leq k$, or "no-instance".

```
1  (G, k), X, S ← BddReduction(G, X, k)
2  if k < 0 then return "no-instance"
3  l ← greedily computed lower bound on the size of a minimum bdd-d-set.
4  if k < l or edge-count test tells "no-instance" then return "no-instance"
5  if maximum degree of G is d then return S
6  Among all vertices of maximum degree, choose a vertex v.
7  if |N(v) ∩ X| > d and |N(v) ∩ X|/|N(v)| > 0.9 then
8      for all size (|N(v) ∩ X| − d)-subsets C ⊆ N(v) ∩ X do
9          call BddSolve (G − C, X \ C, k − |C|)        ▷ Branch on N(v) ∩ X
10     call BddSolve (G − v, X \ {v}, k − 1)                ▷ ... and v.
11 else branch analogously to lines 9–10 on N(v) and v.
12 if all recursive calls of BddSolve returned "no-instance" then
13     return "no-instance"
14 else return S ∪ S', where S' is a smallest set returned by the BddSolve calls.
```

Figure 9.3: Pseudo-code of the basic algorithm to compute a minimum bdd-d-set. The bdd-X-set is a simple greedy solution computed by adding a vertex of highest degree to X until the graph has bounded degree d.

9.3 Implementation and Algorithmic Tricks

Our implementation is written in the functional programming language Objective Caml[3]. A reason for this choice was that we could make use of a purely functional graph data structure. This data structure makes the implementation of a search-tree based algorithm much easier, since we do not have to care about undoing changes to the data structure that were applied in other search tree branches. Moreover, it is a stated (and usually achieved) goal of the Objective Caml developers that Objective Caml code runs at most twice as slow as code generated by a decent C compiler. This speed difference is not a major factor for our considerations, since we are interested in the relative performance of algorithms. Moreover, since we are dealing with exponential-time algorithms, algorithmic improvements usually lead to time savings that cannot be bounded by any constant factor, so this effect seems small in comparison. Our implementation is open source and it is freely available.[4]

Concerning the initial $(d + 2)$-approximate solution X needed for the guided branching, it turns out that a greedy solution, computed by simply taking a vertex of highest degree into the solution until the remaining graph has bounded

[3] See `http://caml.inria.fr/`
[4] `http://theinf1.informatik.uni-jena.de/splex/`

degree d, very often is smaller than a $(d+2)$-approximate solution, although this method does not provably guarantee an approximation factor of $d+2$. Such a greedy solution is computed at the beginning of the computation (before invoking the search tree algorithm), and its size is taken as the initial value of k. Note that our implementation contains many algorithmic tweaks that are not covered by the basic description in Figure 9.3. For instance, the effect of the guided branching can be improved by recomputing X from time to time in the course of the branching process. Moreover, it improves performance significantly if one updates the value of k if a branch has found a solution that is smaller than the initial k. For $d = 1$ (that is, $s = 2$), we implemented the improved branching described in Section 4.4.2 instead of the branching shown in Figure 9.3.

In the following, we comment about some particularities of our search tree implementation. One of the most important issues was the computation of the complement graph, which has to be performed before executing the BddSolve algorithm (Figure 9.3). For sparse graphs, the complement graph is dense and in practice the amount of time and memory to compute it exceeds often the time and memory needed for finding a maximum s-plex. Therefore, we implemented a wrapper that simulates a complement graph, rather than actually computing it. This wrapper, of course, is theoretically slower than the original graph data structure, since the data structure calls have to be translated by the wrapper. However, in practice, this method turns out to be almost always much more efficient than computing the complement graph directly.

For the graphs we considered, it turned out that applying the data reduction rules (see Figure 9.2) in every search tree node yields the best results. In particular, the degree-one rule and the high-degree rule are mostly very effective. To be able to apply these rules more quickly, it seems to be reasonable to implement a data structure that provides fast access to vertices with a particular degree. However, this results in an increase of memory usage, and since the data structure has to be updated very frequently, many operations take more time. For instance, the deletion of a vertex, which is one of the most frequently called routines, needs about twice the time in our experiments. Moreover, we noticed an increased garbage collection overhead. Summarizing, such a data structure slowed down the algorithm; surprisingly, for the degree-one and the high-degree rule a simple sweep over all vertices gave a faster implementation.

Note that we did not implement the iterative compression approach for $d = 1$ (see Theorem 4.5 in Section 4.5) because preliminary experiments just trying all 2^k subsets of a given solution showed that this approach is not yet competitive with a search tree approach in practice. Even enumerating subsets that can yield a solution (excluding subsets that induce a graph of maximum degree $> d$) is not fast enough on the instances we tested. For this reason, we refrained from implementing the compression routine and the whole iterative compression algorithm.

9.4 Experimental Results

All experiments were run on AMD Athlon 64 3700+ machines with 2.2 GHz, 1 M L2 cache, and 3 GB main memory with Debian GNU/Linux 4.0 operating system and the Objective Caml 3.09.2 compiler. The experiments of Balasundaram et al. [BBH09] were performed on Dell Precision PWS690 machines with a 2.66 GHz Xeon Processor, 3 GB main memory, implemented using ILOG CPLEX 10.0. The processor speeds are comparable, so we compare the running times directly without applying a correction factor. The experiments of McClosky and Hicks [MH09] were run on a 2.2 GHz Dual-Core AMD Opteron processor with 3 GB main memory. Note that for both papers [BBH09, MH09] the corresponding source code is not publicly available.

Balasundaram et al. [BBH09] performed experiments with two main groups of graphs. One group can be characterized as social networks, which are derived from real-world data. The second group of graphs contains various graphs using the *Sanchis generator* [SJ96] and clique instances from the second DIMACS challenge [DIM95]. They also performed experiments on two biological networks. Balasundaram et al. [BBH09] used an integer linear programming formulation combined with branch & cut methods. One of their exact algorithms, called BC(MIS), generates cuts based on a greedily computed independent set. They combine this approach with an algorithm that iterates over all vertices and searches an s-plex only in the vicinity of each iterated vertex, combined with a low-degree reduction rule (which corresponds to the high-degree rule in the complement Bounded-Degree-d Vertex Deletion instance). This variant is called *Iterative Peel-Branch-and-Cut (IPBC)* algorithm. In the following, we compare our approach with the BC(MIS) and IPBC algorithms and also with the exact algorithm "OsterPlex" by McClosky and Hicks [MH09], which is an adapted version of an algorithm for finding maximum-cardinality cliques by Östergård [Öst02]. The experiments of McClosky and Hicks [MH09] cover almost all social networks that were analyzed by Balasundaram et al. [BBH09] and the instances from the DIMACS challenge.

9.4.1 Social Networks

This group contains Erdős collaboration networks [GIC07] (ERDŐS graphs), collaboration networks in computational geometry [BM06] (GEOM graphs), and text-mining networks based on Reuters news [BM06] (DAYS graphs). Maximum-cardinality s-plexes are particularly interesting in such graphs; in social network analysis, high density subgraphs play an important role, since they represent, e.g., a group of persons that work closely together. Cliques are too sensible with respect to missing edges, and, even if a group is very closely connected, there might be two persons who did not collaborate, but still can be considered very active members of the group, because they collaborate with nearly every other group member. Other than "high-density subgraph", the s-plex concept is a

clique relaxation with well-defined structural properties [SF78, BBH09].

ERDŐS graphs. Each vertex in an Erdős graph represents a scientist, and two vertices are adjacent if the corresponding scientists have published together. The graphs, obtained from [GIC07], are named "ERDOS-x-y", where x represents the last two digits of the year for which the network was constructed, and y the maximum distance from each vertex to Paul Erdős in the graph. As Balasundaram et al. [BBH09] and McClosky and Hicks [MH09], we consider $x \in \{97, 98, 99\}$ and $y \in \{1, 2\}$.

GEOM graphs. Each vertex represents an author in computational geometry. For each pair of authors the number of joint publications is available. Given a threshold t, two authors are adjacent if they have more than t joint publications. The graphs are constructed from data from Beebe's bibliography page [Bee02] obtained from a computational geometry database [Jon02]. The graphs are obtained from [BM06] and named "GEOM-t", where $t \in \{0, 1, 2\}$ is the threshold.

DAYS graphs. The graphs are based on news released by Reuter during 66 days beginning with the terrorist attacks in New York on September 11, 2001. Each vertex is a selected word that appeared in the news. For each pair of words, the number of sentences where both words appear is available. Given a threshold t, two words are connected by an edge if there exist more than t sentences in which both appear. The graphs, obtained from [BM06], are named "DAYS-t", where $t \in \{3, 4, 5\}$.

These three types of graphs have in common that they are very sparse and show a power-law degree distribution. See Table 9.1 for an overview on the number of vertices, edges, graph density, and maximum-cardinality s-plex sizes (for $1 \leq s \leq 5$) for the ERDŐS, GEOM, and DAYS graphs.

We compared both the IPBC algorithm [BBH09] and the OsterPlex algorithm [MH09] with our methods. We discovered experimentally that the guided branching has a strong effect on the running time for the social network instances, while the BDD-NT-rule and the edge-count rule had only minuscule effects. Therefore, we performed experiments with and without guided branching. The resulting running times for the ERDŐS, GEOM, and DAYS graphs are given in Table 9.2, Table 9.3, and Table 9.4, respectively. For the ERDŐS graphs, our method with guided branching outperforms the approaches by Balasundaram et al. [BBH09] and McClosky and Hicks [MH09] by one or two orders of magnitude. Guided branching is especially very effective for higher values of s. An explanation for this is that since the graphs are very sparse, their complements, on which we solve Bounded-Degree-d Vertex Deletion, are very dense. For this reason, the high-degree rule (Figure 9.2) applies extremely well, and the vertices that are chosen to branch on have mostly a high degree, which makes branching very effective. The guided branching accelerates the search process:

Table 9.1: Number of vertices, edges, edge density, and maximum s-plex sizes for $1 \leq s \leq 5$ for the social networks.

graph	$\|V\|$	$\|E\|$	density	$s=1$	$s=2$	$s=3$	$s=4$	$s=5$
ERDOS-97-1	472	1314	0.01182	7	8	9	11	12
ERDOS-98-1	485	1381	0.01177	7	8	9	11	12
ERDOS-99-1	492	1417	0.01173	7	8	9	11	12
ERDOS-97-2	5488	8972	0.00060	7	8	9	11	12
ERDOS-98-2	5822	9505	0.00056	7	8	9	11	12
ERDOS-99-2	6100	9939	0.00053	8	8	9	11	12
GEOM-0	7343	11898	0.00044	22	22	22	22	22
GEOM-1	7343	3939	0.00015	10	10	11	12	13
GEOM-2	7343	1976	0.00007	8	8	10	11	11
DAYS-3	13332	5616	0.00006	8	10	11	13	13
DAYS-4	13332	3251	0.00004	7	8	9	11	11
DAYS-5	13332	2179	0.00003	7	7	8	10	11
H. Pylori	1570	1399	0.00114	3	5	6	7	8
S. Cerevisiae	2112	2203	0.00099	6	6	7	7	8
S. Pombe	1053	2884	0.00521	8	9	10	11	13

the bdd-d-set X is relatively big (on a dense graph, many vertices have to be put into a bdd-d-set), thus just branching on v and $N(v) \cap X$ instead on v and $N(v)$ (Figure 9.3) still results in a good branching, because $N(v) \cap X$ is not significantly smaller than $N(v)$. Thus, we still have a good branching while having the benefit from the guided branching that our greedy solution X becomes small very quickly. To our surprise, the BDD-NT-rule (almost) does not apply at all. The reason is that X (see "guided branching" in Section 9.2.2) is rather big, and we apply the high-degree rule first (see Figure 9.2), which reduces the graph so effectively that the condition for applying the BDD-rule is (almost) never met. When switching off the high-degree rule, almost all reduction is then performed by the BDD-rule.

For the GEOM graphs, we observe similar speedups of up to two orders of magnitude (see Table 9.3). Interestingly, for some instances our approach does not branch at all; it immediately finds a solution using the data reduction rules. Since the data reduction rules are very effective and few branchings take place, the effect of the guided branching is not as pronounced as for the ERDŐS graphs.

For the DAYS graph, we observe a speedup of up to three orders of magnitude (see Table 9.4) compared to the IPBC algorithm [BBH09]. Note that McClosky and Hicks [MH09] did not include the DAYS graphs in their experiments.

Since the preceding experiments indicate that the running time of our approach does not increase too much with increasing s (recall that $s = d + 1$), we performed experiments on two of the real-world graphs (of medium difficulty) for $1 \leq s \leq 25$. The results are shown in Figure 9.4a. For most values of s, the instances can be solved within some seconds, only very few take several minutes,

Table 9.2: Running times and numbers of search tree nodes for ERDŐS graphs compared with the running times of the IPBC [BBH09] and OsterPlex [MH09] algorithm. Note that our and the OsterPlex experiments were aborted after one hour. Also note that OsterPlex was not tested for $s = 5$.

s	graph	IPBC	OsterPlex	search tree algorithm			
				no guided branching		guided branching	
		seconds	seconds	seconds	nodes	seconds	nodes
2	ERDOS-97-1	1.5	0	0.22	91	0.11	483
	ERDOS-97-2	392.9	1253	8.88	141	4.74	1456
	ERDOS-98-1	1.7	0	0.28	78	0.13	600
	ERDOS-98-2	464.3	1514	8.03	116	6.19	2137
	ERDOS-99-1	1.8	0	0.34	99	0.16	696
	ERDOS-99-2	526.5	1757	9.66	127	7.32	2379
3	ERDOS-97-1	1.8	19	0.41	4422	0.4	4409
	ERDOS-97-2	394.1	≥3600	10.64	30165	10.68	38876
	ERDOS-98-1	1.8	20	0.6	5998	0.61	5998
	ERDOS-98-2	457.1	≥3600	21.15	55389	21	63510
	ERDOS-99-1	1.8	21	0.91	8116	0.89	8256
	ERDOS-99-2	520.0	≥3600	28.18	69522	28.14	94886
4	ERDOS-97-1	2.2	1897	0.58	8949	0.56	8949
	ERDOS-97-2	424.0	≥3600	6.33	25032	6.32	25032
	ERDOS-98-1	2.8	1675	0.6	8840	0.58	8840
	ERDOS-98-2	614.7	≥3600	8.63	33360	8.64	33360
	ERDOS-99-1	1.8	1783	0.9	12793	0.92	12793
	ERDOS-99-2	526.3	≥3600	16.39	72695	16.36	72256
5	ERDOS-97-1	5.7	–	16.32	177845	2.83	42459
	ERDOS-97-2	1042.8	–	1462.86	4948746	25.4	172281
	ERDOS-98-1	7.9	–	37.76	347168	2.78	42169
	ERDOS-98-2	1664.6	–	≥3600	8172145	38.27	245845
	ERDOS-99-1	9.9	–	91.72	697284	4.98	68172
	ERDOS-99-2	653.5	–	≥3600	7148308	112.44	635655

and exactly one takes almost one hour. We also observe in Figure 9.4b that the size of a maximum s-plex increases almost linearly with the value of s. We conclude that our approach seems to be able to find maximum s-plexes for a wide range of the parameter s for these types of graph.

9.4.2 Biological Networks

Balasundaram et al. [BBH09] performed experiments on two biological networks, namely protein-protein interaction networks of *H. Pylori* and *S. Cerevisiae*. In these graphs, vertices represent proteins and edges indicate that the pair of proteins forming the endpoints are known to interact. In such protein-protein interaction networks, s-plexes correspond to functional modules (see, e.g, [BBT05]).

Table 9.3: Running times and numbers of search tree nodes for GEOM graphs.

s	graph	IPBC	OsterPlex	search tree algorithm			
				no guided branching		guided branching	
		seconds	seconds	seconds	nodes	seconds	nodes
2	GEOM-0	2384.4	397	10.44	0	10.47	0
	GEOM-1	753.2	1118	5.53	14	5.67	108
	GEOM-2	530.6	1145	3.82	13	3.77	115
3	GEOM-0	2387.1	≥3600	10.44	0	10.39	0
	GEOM-1	747.7	≥3600	6.42	7472	5.65	885
	GEOM-2	524.3	≥3600	3.8	1	3.82	1
4	GEOM-0	2383.7	≥3600	10.44	0	10.4	0
	GEOM-1	743.7	≥3600	5.84	5021	5.8	4871
	GEOM-2	522.2	≥3600	3.77	1	3.7	1
5	GEOM-0	2298.1	–	10.47	0	10.53	0
	GEOM-1	691.6	–	7.01	21952	6.94	21952
	GEOM-2	472.6	–	7.78	58445	7.88	58445

We add one additional network to the set of instances, namely the protein-protein interaction network of *S. Pombe* (fission yeast). We used this network in our initial example in this chapter (Figure 9.1). The graph of S. Pombe was generated using data from the BioGRID database (`http://www.thebiogrid.org`). See Table 9.1 for the number of vertices and edges, density, and maximum s-plex sizes for $1 \leq s \leq 5$ for these biological networks. For the biological instances, we observe a very different behavior of our approach depending on the network (Table 9.5). For S. Cerevisiae and S. Pombe our algorithm performs very good for all considered values of s. For S. Cerevisiae, it is about two orders of magnitude faster than the IPBC algorithm by Balasundaram et al. [BBH09]. However, for H. Pylori the running time of our approach increases very quickly with increasing s and is much slower than the IPBC algorithm. An explanation for that behavior is that the high-degree rule cannot reduce the graph sufficiently; therefore, the parameter k is still rather large when the algorithm starts branching and therefore the search space explodes. It would be interesting to see why IPBC performs so extremely well on this instance compared to our approach; this could help to combine the "best of both worlds" into one algorithm.

9.4.3 Sanchis and DIMACS Graphs

The second group of graphs considered by Balasundaram et al. [BBH09] contains various graphs using the *Sanchis generator* [SJ96] and clique instances from the second DIMACS challenge [DIM95]. The Sanchis generator [SJ96] produces graphs with known maximum clique size with a specified number of vertices n and edges m, and a construction parameter r. As Balasundaram et al. [BBH09], we fixed the maximum clique size at $\lceil n/5 \rceil$, and the construction pa-

Table 9.4: Running time and number of search tree nodes for DAYS graphs.

s	graph	IPBC	search tree algorithm			
			no guided branching		guided branching	
		seconds	seconds	nodes	seconds	nodes
2	DAYS-3	3367.8	22.01	10	22.12	256
	DAYS-4	2635.7	16.28	12	16.46	180
	DAYS-5	2462.9	0.11	16	0.1	179
3	DAYS-3	3395.4	63.5	119666	22.84	4396
	DAYS-4	3395.4	16.47	2282	16.44	2282
	DAYS-5	2445.5	0.27	2302	0.27	2302
4	DAYS-3	3489.8	21.79	1	21.98	1
	DAYS-4	3642.3	16.29	1	16.41	1
	DAYS-5	2426.3	92.89	550125	0.19	37
5	DAYS-3	15336.9	79.06	423511	79.2	423511
	DAYS-4	6201.4	28.93	149970	28.66	149970
	DAYS-5	2820.8	$\geq$3600	6092198	2.38	21423

rameter to $\lfloor 0.75(n/c-1)\rfloor$. The number of edges is determined by the density d, that is, we compute the number of edges as $m := \lfloor dn(n-1)/2\rfloor$. We performed experiments for $n \in \{100, 200\}$ and $d \in \{0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$.

Balasundaram et al. [BBH09] used Sanchis graphs to study how the efficiency of their methods depends on the number of graph vertices, the density of the graph, and on the value s defining s-plexes. Their methods perform best on sparse graphs, and become less effective on dense graphs. Likewise, small graphs can be solved quickly, while larger graphs become more difficult to solve. Balasundaram et al. [BBH09] performed experiments with the BC(MIS) algorithm for $s \in \{1, 2\}$. They observed that the case $s = 2$ is generally more difficult to solve than $s = 1$.

We observe the same general behavior as for the BC(MIS) algorithm, that is, dense Sanchis graphs are harder to solve than sparse ones, and graphs with many vertices are harder to solve than graphs with few vertices. We can observe that, especially on sparse instances, our approach is slightly slower than the BC(MIS) algorithm (see Figure 9.5a). However, the available data seems to indicate that the running time of our approach increases not as quickly with increasing density as the BC(MIS) algorithm does. The described effects are more pronounced for higher values of n, the number of vertices. For instance, for $n = 200$ our approach is about one order of magnitude slower than BC(MIS) for low density (e.g., $d = 0.4$), but becomes faster than BC(MIS) around $d = 0.7$. We studied also how the running time depends on n for fixed density (see Figure 9.5b). Our approach is generally slower than BC(MIS) for increasing n, and the speed difference is higher for lower density values. The most significant difference to the social networks is that the Sanchis graphs are more dense, and therefore the complement graph is more sparse and the high-degree rule does not apply so often, which makes our approach less efficient. Interestingly, the BDD-NT-Rule

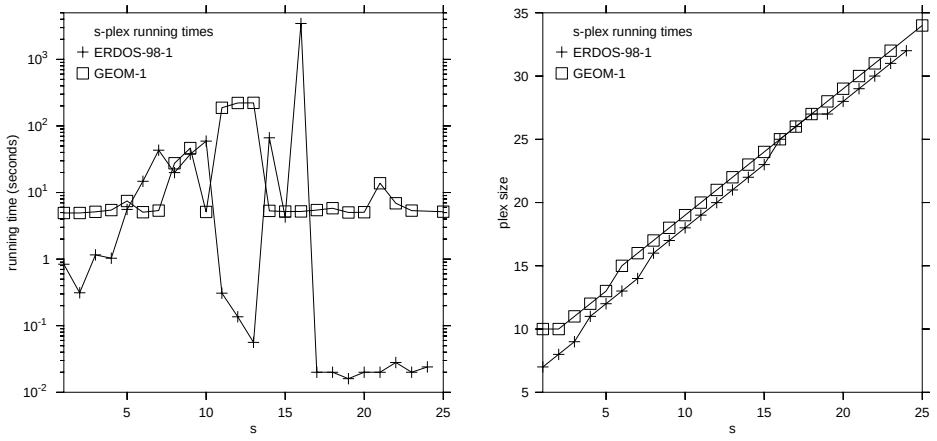


Figure 9.4: (a) Running times of our approach and (b) sizes of the resulting maximal s-plexes for $1 \leq d \leq 25$ on ERDOS-98-1 and GEOM-1 graphs. Missing data points are due to the exceeded running time limit of 60 minutes.

is applied more often in Sanchis graphs compared to the social networks. Note that in general guided branching shows no effect in Sanchis graphs.

Finally, we briefly report about our findings concerning instances from the DIMACS challenge. We compare with the BC(MIS) algorithm [BBH09] and the OsterPlex algorithm [MH09]. The results, which cover all instances that are used by Balasundaram et al. [BBH09], are shown in Table 9.6. Summarizing, out of the 32 considered instances we could solve 25 instances for $s = 1$ and 17 instances for $s = 2$, while BC(MIS) could solve 20 instances for $s = 1$ and 16 instances for $s = 2$ within a running time limit of three hours. Compared to the OsterPlex algorithm, we could solve within one hour all but five instances for $s = 2$, which OsterPlex can solve within that time. Summarizing, BC(MIS) is comparable with our approach, and OsterPlex is at least as good as our approach for these instances. In general, "hard" instances cannot be solved efficiently by either of the three compared algorithms and "easy" instances are solved quickly by all the three algorithms, but there are a few exceptions where one method seems to outperform the others. In this respect, it would be interesting to study whether the OsterPlex and the BC(MIS) algorithms could be efficiently combined with ours.

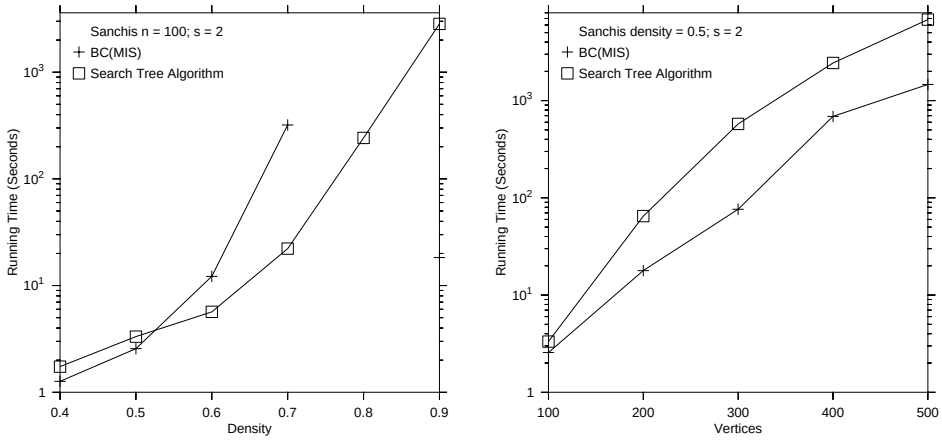
9.5 Further Remarks

Effectiveness of the Data Reduction Rules. The three data reduction rules (Figure 9.2) behave very differently in our experiments. The simple high-degree rule is the most effective and time-efficient data reduction rule, although in theory it does not yield the best-possible problem kernel available. We recommend

Table 9.5: Running time and number of search tree nodes for the biological networks.

s	graph	search tree algorithm				
		IPBC	no guided branching		guided branching	
		seconds	seconds	nodes	seconds	nodes
1	H. Pylori	11.5	5.22	357	4.62	405
	S. Cerevisiae	44.1	0.29	0	0.29	0
	S. Pombe	-	0.56	45	1.07	178
2	H. Pylori	12.6	11.24	289	4.48	5118
	S. Cerevisiae	46.4	0.29	1	0.29	1
	S. Pombe	-	1.00	28	1.06	1489
3	H. Pylori	37.8	43.59	233060	42.62	219782
	S. Cerevisiae	26.6	0.29	1	0.29	1
	S. Pombe	-	13.87	50127	4.27	24641
4	H. Pylori	29.3	$\geq$3600	$\geq$7494976	1580.7	7249448
	S. Cerevisiae	45.0	0.31	23	0.31	23
	S. Pombe	-	20.55	126160	21.03	126160
5	H. Pylori	133.1	$\geq$3600	$\geq$14364619	$\geq$3600	$\geq$17201945
	S. Cerevisiae	41.8	249.23	1224879	1.83	6532
	S. Pombe	-	0.82	1	0.84	1

to apply it in every search tree node, especially in dense Bounded-Degree-d Vertex Deletion instances. The degree-one rule is less often applied, but still it is quite effective on some instances. Concerning the BDD-NT-rule, it is also applied less often than the high-degree rule. We repeated some of the above experiments with the high-degree rule and the degree-one rule disabled. Then, the applications of the BDD-NT-rule increase dramatically. However, we observed that the running time with and without BDD-NT-rule is approximately the same for most of the instances. We observed two reasons for this behavior: one reason is the running time of the BDD-NT-rule; in the present version, it seems to be still too slow to be used to effectively speed up the search tree algorithm. A bottleneck in the BDD-NT-rule is the computation of a maximum flow (cf. Section 4.3), which we implemented using a simple augmenting path computation; however, this method turned out to be still faster than using an existing (experimental) maximum flow library for Objective Caml. A more sophisticated routine to compute maximum flows could significantly speed up the search process. The second reason is that the greedy solution X is rather big in almost all the tested instances. In many cases, the condition $|N(X)| > (d+1)|X|$ (cf. Figure 9.2) is satisfied only in search tree nodes that are very close to the leaves of the search tree. The input instance for such search tree nodes often contains only a few vertices, and the instance is solved very efficiently within a few branching steps. The benefit of reducing such small instances before branching has no big influence, compared to the application of, e.g., the high-degree rule on the input graph even



(a) (b)

Figure 9.5: Running times of our approach (search tree algorithm) compared with the running times of the BC(MIS) approach by Balasundaram et al. [BBH09].

before starting to branch.

Since data reduction rules are in some sense universal (that is, they can be always applied before solving an instance with virtually any method), it makes sense to combine the data reduction rules presented in this chapter with the BC(MIS) and the OsterPlex algorithm. We think that this is an interesting topic for future research.

Related Experimental Work. In order to obtain algorithms for Maximum s-Plex that are faster than the ones compared in this chapter, it might be necessary to add further restrictions such as the one of "isolation" [II09, KHMN09, HKMN09b]; this concept restricts the number of edges between a dense subgraph (e.g., clique or s-plex) and vertices outside of the dense subgraph. Moreover, in practice it is also of interest to find not only one maximum s-plex, but to list several alternative "large" s-plexes. This leads to the task of s-plex *enumeration*, that is, to enumerate all maximal s-plexes in a given graph. In order to speed up the enumeration, it can be combined with isolation concepts. We performed experiments for five different variants of such isolation concepts for the task of enumerating cliques. We showed that such isolation concepts can help to speed up the enumeration significantly and to filter out cliques with particularly interesting properties. See Komusiewicz et al. [KHMN09] for more details.

9.6 Outlook

In some analogy to previous work on maximum-cardinality clique finding [ACF+04, AFLS07, CLS+05], we demonstrated that a fixed-parameter approach provides competitive algorithms for finding maximum-cardinality s-plexes. Clearly, due

to the NP-hardness of the problem, there are limitations concerning the range of practical feasibility. On the one hand, we believe that there is still some room for further tuning our algorithms and implementations (which in future work also should be compared with other approaches in an experimental study that is based on the *same* platform); on the other hand, we think that at some point more restrictions such as the one of "isolation" (see [II09, KHMN09, HKMN09b]) have to be imposed in order to gain practical algorithms. Our focus was on finding s-plexes of maximum size; studies concerning efficient approximation algorithms are left open.

It is conceivable that the algorithm presented in this chapter can be converted into an enumeration algorithm if one does not use the BDD-NT-Rule and the Degree-One Rule. Since the whole search space has to be traversed, it seems also clear that the guided branching heuristic would not show effect. Hence, isolation concepts might be necessary in order to derive efficient enumeration algorithms, also because for $s \geq 2$ the search space for Maximum s-Plex is larger than the search space for Maximum Clique (for which the isolation concepts were originally designed).

Table 9.6: Table showing the running times of our algorithm and the corresponding s-plex sizes of DIMACS instances. A "B" superscript means that Balasundaram et al. [BBH09] solved the corresponding instance to optimality within three hours, but our algorithm did not terminate within that time. A "b" means that we solved the corresponding instance to optimality within three hours, but the algorithm of Balasundaram et al. [BBH09] did not terminate within that time. Similarly, an "M" means that McClosky and Hicks [MH09] solved the corresponding instance to optimality within one hour, but our algorithm did not terminate within that time. If our algorithm did not terminate within the running time limit, then we state the lower bound x and upper bound y of the maximum s-plex size that could be computed in the given time as an interval $[x, y]$.

graph	$\|V\|$	density	1-plex size	seconds	2-plex size	seconds
c-fat200-1	200	0.077	12	0.21	12	1.00
c-fat200-2	200	0.163	24	0.43	24	3.42
c-fat200-5	200	0.426	58	1.20	58	20.35
c-fat500-1	500	0.036	14	3.94	14	11.01
c-fat500-2	500	0.073	26	7.43	26	48.31
c-fat500-5	500	0.186	64	18.31	64	331.65
c-fat500-10	500	0.374	126	37.68	126	1483.22
hamming6-2	64	0.905	32	0.00	32	1.27
hamming6-4	64	0.349	4	0.05	6	0.26
hamming8-2	256	0.969	128	0.04	$[128,192]^{BM}$	> 10800
hamming8-4	256	0.639	16	275.60	$[16,171]^{BM}$	> 10800
hamming10-2	1024	0.990	512	0.8	$[512,768]^{M}$	> 10800
hamming10-4	1024	0.829	[30,512]	> 10800	[36,683]	> 10800
johnson8-2-4	28	0.556	4	0.00	5	0.02
johnson8-4-4	70	0.768	14	0.47	14	30.13
MANN_a9	45	0.927	16	0.00	26	0.12
MANN_a27	45	0.927	126	2.98	236	4053.34
MANN_a45	1035	0.996	345^{b}	583.66	[662,697]	> 10800
keller4	171	0.649	11	23.51	15^{M}	3677.89
brock200_1	200	0.745	21^{b}	885.30	[24,134]	> 10800
brock200_2	200	0.496	12	25.61	13^{b}	558.95
brock200_4	200	0.658	17	226.10	20^{b}	8246.32
brock400_2	400	0.749	[23,200]	> 10800	[26,267]	> 10800
brock400_4	400	0.749	[24,200]	> 10800	[26,267]	> 10800
brock800_2	800	0.651	[19,400]	> 10800	[21,534]	> 10800
brock800_4	800	0.650	[19,400]	> 10800	[22,534]	> 10800
p_hat300-1	300	0.244	8	30.75	10^{b}	498.61
p_hat300-2	300	0.489	25^{b}	266.94	[30,200]	> 10800
p_hat300-3	300	0.744	36^{b}	9410.00	[39,200]	> 10800
p_hat700-1	700	0.249	11^{b}	1624.02	$[11,467]^{M}$	> 10800
p_hat700-2	700	0.498	[40,350]	> 10800	[47,467]	> 10800
p_hat700-3	700	0.748	[58,350]	> 10800	[69,467]	> 10800

Chapter 10

Outlook

This thesis studies various covering/vertex deletion and generalized matching problems. We recapitulate the findings presented in Chapters 4–9, the main chapters of this work, and then outline some possible future research directions.

The main finding in Chapter 4 is a generalization of the classical Nemhauser-Trotter theorem for Vertex Cover to the problem of obtaining a graph of bounded degree by a minimum number of vertex deletions. This result also yields a problem kernel of $O(k^{1+\epsilon})$ vertices for any constant $\epsilon > 0$. The central idea behind this quite technical and involved proof is to use a constant-factor approximate solution as starting point and then to employ an iterated search approach based on maximum flow techniques. Furthermore, Chapter 4 includes a simple kernelization result, fixed-parameter algorithms, and a hardness result.

In Chapter 5 we showed a problem kernel for the problem of making a graph regular by a minimum number of vertex deletions. The basic technique is a rather involved gadget construction that is used to replace big regular parts of the graph by smaller ones.

Chapter 6 analyzes the computational complexity of the task of finding a disjoint solution for a large class of vertex deletion problems. This task is centrally occurring in most known applications of iterative compression. The main technique to obtain the polynomial-time solvable results is reduction to a matching problem, and the technique for showing NP-hardness is based on a modified version of a known hardness framework for vertex deletion problems and further hardness proofs for particular cases where the framework fails.

In Chapter 7 we considered the problem of packing a fixed graph into a given graph, and give problem kernelization results. These are based on a special kind of packing with allowed overlaps together with matching techniques.

In Chapter 8 we considered the problem of packing edges with pairwise distance at least two into a graph. The main results were obtained with region decomposition and dynamic programming techniques.

Finally, Chapter 9 reports about implementation and experiments for finding a particular kind of dense graph in a given graph. The algorithm is based

on a transformation to the Bounded-Degree-d Vertex Deletion problem considered in Chapter 4.

The main techniques for fixed-parameter algorithms used in this paper were bounded search trees, problem kernelization, and iterative compression [HNW08]. While we used bounded search trees and problem kernelization to obtain practical algorithms for Bounded-Degree-d Vertex Deletion, we observed that the theoretically fastest iterative compression approach for Bounded-Degree-1 Vertex Deletion does not yield fast algorithms in practice on the considered test instances. In contrast, iterative compression combined with data reduction yields fast practical algorithms for many feedback set problems [BHTW09, Hüf07, Hüf09, HBN09, HNW08], for which no other fast parameterized algorithms (e.g., based on bounded search trees) are available. One of the reasons that iterative compression was not the fastest method in our case it the size of the parameter. In most of the instances which we could solve very quickly, the parameter is rather big, often only slightly smaller than the number of vertices in the graph. In the case of the iterative compression technique, this slows down the algorithm drastically. Another observation is that the problem kernel size, which is currently the focus in research on problem kernelization, is not necessarily a decisive measure for its practical success. As we have seen in this thesis, a faster problem kernelization with worse size bound can be much more efficient. Moreover, it can speed up a kernelization significantly if one first applies a fast problem kernelization with worse size bounds, and afterwards a slower problem kernelization with better size bounds, which then runs faster compared to its direct application on the input instance ("cascading effect"). We therefore advocate that research should also strive for fast problem kernelizations, maybe even for problem kernels that do not match the currently best-known size bound.

In the conclusions of each chapter, we already pointed out some future research directions for each particular problem and will not repeat them here. Rather, we outline two more general topics that might be interesting for future research.

Kernelization Duality. There are quite a few parameterized problems for which one can observe some kind of duality with respect to problem kernelization between covering and matching problems. For instance, the technique for our $O(k)$-vertex problem kernel for Bounded-Degree-1 Vertex Deletion can be used almost directly for the problem of packing at least k vertex-disjoint copies of a non-induced P_3 (cf. Section 4.3). The point is that the forbidden substructure for Bounded-Degree-1 Vertex Deletion is exactly the structure one aims to pack in the packing problem. Other examples are

- the problem of deleting at most k vertices in order to make a graph triangle-free [Abu09] and the problem of packing k vertex-disjoint triangles into a graph [FHR$^+$04],

- the problem of deleting at most k edges in order to make a graph triangle-

free [BKM09] and the problem of packing k *edge*-disjoint triangles into a graph [MPS04], and

- Hitting Set [Abu09] and Set Packing [AK09] / H-Packing (cf. Chapter 7).

While the kernelization algorithms for each pair of problems are very similar, each pair of problems behaves quite differently with respect to parameterized algorithms. The best-known parameterized algorithms for the covering problems often use bounded search tree approaches or also iterative compression [GMN09], while the dominating technique for the matching/packing problems is color-coding [AYZ95], sometimes additionally combined with divide-and-conquer strategies [CKL+09]. The results stated above indicate that the investigation of such "kernelization dualities" might be a worthwhile research topic. For instance, concrete problem pairs to start with could be

- Undirected Feedback Vertex Set with an $O(k^2)$-vertex problem kernel [Tho09] and Disjoint Cycle Packing with an $O(k^2 \log^2 k)$-vertex problem kernel [BTY09], and
- the problem of deleting at most k vertices from a graph such that it contains no induced path of constant length s and the problem of packing an induced path of length s into a given graph (cf. [HR97, MT05]).

Furthermore, can similar dual kernelization relations be observed for problems that are neither covering nor matching problems?

Hybrid (Parameterized) Algorithms. Although there already is some experience for algorithm engineering with a focus on parameterized algorithms (see, e.g., [ACF+04, BHTW09, CLS+05, FWY09, HBN09, Hüf07, Hüf09]), this field is still in its infancy and has probably a lot of potential. Most of the algorithm engineering efforts so far seem to be focusing on one particular (natural) parameter of the considered problem. We propose that future algorithm engineering projects should also consider *hybrid* approaches, that is, that the algorithm analyzes the input instance before starting to solve it and tries to choose a parameter and corresponding solving method that seems to be most appropriate. For instance, for graph problems a very simple example of such an approach could be the following. First, use a heuristic to compute a tree decomposition of the input graph; if the width of this tree decomposition is small, then try to use a dynamic programming approach. Otherwise, proceed with other (structural) parameters and parameterized algorithms. For instance, such (structural) parameters could be

- maximum vertex degree,
- solution size,

- density,
- girth ("small cycles make problems hard" [RS08]), and many more.

In this process, also "non-obvious" parameters should be taken into account. For example, the "dual parameters" that have been used for Maximum Clique ([ACF+04]) or Maximum s-Plex (Chapter 9) could be considered "non-obvious". Moreover, also non-parameterized approaches like integer linear programming should be added to the pool of possible techniques. Obviously, such an "all-embracing" approach would be far more complex than existing projects, and the work probably cannot be done by one single research group. Therefore, the development of publicly available software libraries (such as heuristic and exact algorithms for computing tree decompositions, see `http://www.treewidth.com/`) and data collections is particularly important for the advance of (parameterized) algorithm engineering in the future.

Bibliography

[ABF+02] 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 p. 150.

[Abu09] Faisal N. Abu-Khzam. A kernelization algorithm for d-Hitting Set. *Journal of Computer and System Sciences*, 2009. Available electronically. Cited on pp. 113, 121, 127, 178, and 179.

[ABWB+09] Mary V. Ashley, Tanya Y. Berger-Wolf, Piotr Berman, Wanpracha Art Chaovalitwongse, Bhaskar DasGupta, and Ming-Yang Kao. On approximating four covering and packing problems. *Journal of Computer and System Sciences*, 75(5):287–302, 2009. Cited on pp. 3, 111, 112, and 131.

[ACF+04] Faisal N. Abu-Khzam, Rebecca L. Collins, Michael R. Fellows, Michael A. Langston, W. Henry Suters, and Christopher T. Symons. Kernelization algorithms for the Vertex Cover problem: Theory and experiments. In *Proceedings of the 6th Workshop on Algorithm Engineering and Experiments (ALENEX '04)*, pages 62–69. ACM/SIAM, 2004. Cited on pp. 2, 30, 85, 156, 174, 179, and 180.

[ACG+99] 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 p. 4.

[ACP87] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic and Discrete Methods*, 8(2):277–284, 1987. Cited on p. 16.

[AF06] Faisal N. Abu-Khzam and Henning Fernau. Kernels: Annotated, proper and induced. In *Proceedings of the 2nd International Work-*

shop on Parameterized and Exact Computation (IWPEC '06), volume 4169 of *Lecture Notes in Computer Science*, pages 264–275. Springer, 2006. Cited on p. 127.

[AFLS07] Faisal N. Abu-Khzam, Michael R. Fellows, Michael A. Langston, and W. Henry Suters. Crown structures for vertex cover kernelization. *Theory of Computing Systems*, 41(3):411–430, 2007. Cited on pp. 2, 11, 26, 30, 42, 156, and 174.

[AFN04] 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 pp. 141, 142, 143, and 145.

[AK09] Faisal N. Abu-Khzam. A quadratic kernel for 3-set packing. 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 81–87. Springer, 2009. Cited on pp. 131 and 179.

[ALSS06] Faisal N. Abu-Khzam, Michael A. Langston, Pushkar Shanbhag, and Christopher T. Symons. Scalable parallel algorithms for FPT problems. *Algorithmica*, 45(3):269–284, 2006. Cited on pp. 2, 12, and 13.

[AN02] Jochen Alber and Rolf Niedermeier. Improved tree decomposition based algorithms for domination-like problems. In *Proceedings of the 5th Latin American Symposium on Theoretical Informatics (LATIN '02)*, volume 2286 of *Lecture Notes in Computer Science*, pages 613–628. Springer, 2002. Cited on p. 150.

[ARS02] James Abello, Mauricio G. C. Resende, and Sandra Sudarsky. Massive quasi-clique detection. In *Proceedings of the 5th Latin American Symposium on Theoretical Informatics (LATIN '02)*, volume 2286 of *Lecture Notes in Computer Science*, pages 598–612. Springer, 2002. Cited on p. 90.

[AYZ95] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, 1995. Cited on p. 179.

[AYZ97] Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997. Cited on p. 114.

[Bak94] Brenda S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM*, 41(1):153–180, 1994. Cited on p. 112.

[BBBT09] Sebastian Böcker, Sebastian Briesemeister, Quang Bao Anh Bui, and Anke Truß. Going weighted: Parameterized algorithms for cluster editing. *Theoretical Computer Science*, 410(52):5467–5480, 2009. Cited on p. 13.

[BBH09] Balabhaskar Balasundaram, Sergiy Butenko, and Illya V. Hicks. Clique relaxations in social network analysis: The maximum k-plex problem. *Operations Research*, 2009. To appear. Cited on pp. 26, 155, 156, 157, 158, 166, 167, 168, 169, 170, 171, 172, 174, and 176.

[BBK+04] Hari Balakrishnan, Christopher L. Barrett, V. S. Anil Kumar, Madhav V. Marathe, and Shripad Thite. The distance-2 matching problem and its relationship to the MAC-layer capacity of ad hoc wireless networks. *IEEE Journal on Selected Areas in Communications*, 22(6):1069–1079, 2004. Cited on pp. 3, 135, and 136.

[BBK09] Sebastian Böcker, Sebastian Briesemeister, and Gunnar W. Klau. Exact algorithms for cluster editing: Evaluation and experiments. *Algorithmica*, 2009. Available electronically. Cited on pp. 12 and 13.

[BBP05] Vladimir Boginski, Sergiy Butenko, and Panos M. Pardalos. Statistical analysis of financial networks. *Computational Statistics & Data Analysis*, 48(2):431–443, 2005. Cited on p. 1.

[BBP06] Vladimir Boginski, Sergiy Butenko, and Panos M. Pardalos. Mining market data: A network approach. *Computers & Operations Research*, 33(11):3171–3184, 2006. Cited on p. 1.

[BBT05] Balabhaskar Balasundaram, Sergiy Butenko, and Svyatoslav Trukhanovzu. Novel approaches for analyzing biological networks. *Journal of Combinatorial Optimization*, 10(1):23–39, 2005. Cited on pp. 157, 158, and 169.

[BBYG00] 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 p. 20.

[BCK+05] Nicole E. Baldwin, Elissa J. Chesler, Stefan Kirov, Michael A. Langston, Jay R. Snoddy, Robert W. Williams, and Bing Zhang. Computational, integrative, and comparative methods for the elucidation of genetic coexpression networks. *Journal of Biomedicine and Biotechnology*, 2005(2):172–180, 2005. Cited on p. 26.

[BCL04] Rodica Boliac, Kathie Cameron, and Vadim V. Lozin. On computing the dissociation number and the induced matching number of bipartite graphs. *Ars Combinatoria*, 72:241–253, 2004. Cited on p. 27.

[BDFH09] 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 p. 113.

[BECF+06] 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 p. 20.

[Bee02] Nelson H.F. Beebe. Nelson H.F. Beebe's bibliographies page. `http://www.math.utah.edu/~beebe/bibliographies.html`, 2002. Cited on p. 167.

[BFL+] 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)*. IEEE. Cited on p. 141.

[BG93] Jonathan F. Buss and Judy Goldsmith. Nondeterminism within P. *SIAM Journal on Computing*, 22(3):560–572, 1993. Cited on p. 28.

[BHMW08] Hans-Joachim Böckenhauer, Juraj Hromkovič, Tobias Mömke, and Peter Widmayer. On the hardness of reoptimization. In *Proceedings of the 34th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM '08)*, volume 4910 of *Lecture Notes in Computer Science*, pages 50–65. Springer, 2008. Cited on p. 91.

[BHR08] Richard C. Brewster, Pavol Hell, and Romeo Rizzi. Oriented star packings. *Journal of Combinatorial Theory. Series B*, 98(3):558–576, 2008. Cited on p. 23.

[BHTW09] Sebastian Böcker, Falk Hüffner, Anke Truß, and Magnus Wahlström. A faster fixed-parameter approach to drawing binary tanglegrams. In *Proceedings of the 4th International Workshop on Parameterized and Exact Computation (IWPEC '09)*, volume 5917 of *Lecture Notes in Computer Science*, pages 38–49. Springer, 2009. Cited on pp. 178 and 179.

[Big94] Norman Biggs. *Algebraic Graph Theory*. Cambridge University Press, 2nd edition, 1994. Cited on p. 65.

[BIL08] Vincenzo Bonifaci, Ugo Di Iorio, and Luigi Laura. The complexity of uniform Nash equilibria and related regular subgraph problems. *Theoretical Computer Science*, 401(1-3):144–152, 2008. Cited on pp. 64 and 66.

[BJL+90] Francine Berman, David S. Johnson, Frank Thomson Leighton, Peter W. Shor, and Larry Snyder. Generalized planar matching. *Journal of Algorithms*, 11(2):153–184, 1990. Cited on p. 112.

[BK08] Hans L. Bodlaender and Arie M. C. A. Koster. Combinatorial optimization on graphs of bounded treewidth. *The Computer Journal*, 51(3):255–269, 2008. Cited on p. 16.

[BKM09] Daniel Brügmann, Christian Komusiewicz, and Hannes Moser. On generating triangle-free graphs. *Electronic Notes in Discrete Mathematics*, 32:51–58, 2009. Cited on pp. 179 and 211.

[BLS99] Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph Classes: a Survey*, volume 3 of *SIAM Monographs on Discrete Mathematics and Applications*. Society for Industrial and Applied Mathematics, 1999. Cited on p. 135.

[BM06] Vladimir Batagelj and Andrej Mrvar. Pajek datasets. `http://vlado.fmf.uni-lj.si/pub/networks/data/`, 2006. Accessed January 2009. Cited on pp. 166 and 167.

[Bod94] Hans L. Bodlaender. On disjoint cycles. *International Journal of Foundations of Computer Science*, 5:59–68, 1994. Cited on p. 20.

[Bod96] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996. Cited on p. 16.

[Bod06] Hans L. Bodlaender. Treewidth: Characterizations, applications, and computations. In *Proceedings of the 32nd International Workshop on Graph-Theoretic Conecpts in Computer Science (WG '06)*, volume 4271 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2006. Cited on p. 14.

[Bod07] Hans L. Bodlaender. A cubic kernel for feedback vertex set. In *Proceedings of the 24th International Symposium on Theoretical Aspects of Computer Science (STACS '07)*, volume 4393 of *Lecture Notes in Computer Science*, pages 320–331. Springer, 2007. Cited on pp. 12 and 20.

[BTY09] 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 p. 179.

[BW06] Sergiy Butenko and W. E. Wilhelm. Clique-detection models in computational biochemistry and genomics. *European Journal of Operational Research*, 173(1):1–17, 2006. Cited on p. 1.

[BYE85] Reuven Bar-Yehuda and Shimon Even. A local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Discrete Mathematics*, 25:27–45, 1985. Cited on pp. 1, 29, and 30.

[BYRH09] Reuven Bar-Yehuda, Dror Rawitz, and Danny Hermelin. An extension of the Nemhauser & Trotter theorem to generalized vertex cover with applications. In *Proceedings of the 7th Workshop on Approximation and Online Algorithms (WAOA '09)*, Lecture Notes in Computer Science. Springer, 2009. To appear. Cited on p. 30.

[Cai96] Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58(4):171–176, 1996. Cited on pp. 20, 64, 84, and 127.

[Cam89] Kathie Cameron. Induced matchings. *Discrete Applied Mathematics*, 24:97–102, 1989. Cited on p. 136.

[Cam04] Kathie Cameron. Induced matchings in intersection graphs. *Discrete Mathematics*, 278(1–3):1–9, 2004. Cited on p. 136.

[Cam09] Kathie Cameron. Brambles and independent packings in chordal graphs. *Discrete Mathematics*, 309(18):5766–5769, 2009. Cited on pp. 23 and 154.

[CC90] F. Cheah and Derek G. Corneil. The complexity of regular subgraph recognition. *Discrete Applied Mathematics*, 27(1-2):59–68, 1990. Cited on p. 66.

[CC04] Miroslav Chlebík and Janka Chlebíková. Approximation hardness of dominating set problems. In *Proceedings of the 12th Annual European Symposium on Algorithms (ESA '04)*, volume 3221 of *Lecture Notes in Computer Science*, pages 192–203. Springer, 2004. Cited on p. 135.

[CC06] Miroslav Chlebík and Janka Chlebíková. Complexity of approximating bounded variants of optimization problems. *Theoretical Computer Science*, 354(3):320–338, 2006. Cited on p. 112.

[CC08] Miroslav Chlebík and Janka Chlebíková. Crown reductions for the minimum weighted vertex cover problem. *Discrete Applied Mathematics*, 156:292–312, 2008. Cited on p. 30.

[CCDF97] 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 p. 11.

[CFJ04] 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 Conecpts in Computer Science (WG '04)*, volume 3353 of *Lecture Notes in Computer Science*, pages 257–269. Springer, 2004. Cited on p. 42.

[CFKX07] 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 pp. 113, 141, and 142.

[CFL+08] Jianer Chen, Fedor V. Fomin, Yang Liu, Songjian Lu, and Yngve Villanger. Improved algorithms for feedback vertex set problems. *Journal of Computer and System Sciences*, 74(7):1188–1198, 2008. Cited on pp. 14, 20, 21, 84, 85, 89, 90, and 91.

[CFST79] Vašek Chvátal, Herbert Fleischner, John Sheehan, and Carsten Thomassen. Three-regular subgraphs of four-regular graphs. *Journal of Graph Theory*, 3(4):371–386, 1979. Cited on p. 66.

[CH06] Kathie Cameron and Pavol Hell. Independent packings in structured graphs. *Mathematical Programming*, 105(2-3):201–213, 2006. Cited on pp. 23 and 154.

[Cha03] Jou-Ming Chang. Induced matchings in asteroidal triple-free graphs. *Discrete Applied Mathematics*, 132(1-3):67–78, 2003. Cited on p. 136.

[CKJ01] Jianer Chen, Iyad A. Kanj, and Weijia Jia. Vertex cover: Further observations and further improvements. *Journal of Algorithms*, 41(2):280–301, 2001. Cited on pp. 11, 21, and 30.

[CKL07] Domingos M. Cardoso, Marcin Kaminski, and Vadim V. Lozin. Maximum r-regular induced subgraphs. *Journal of Combinatorial Optimization*, 14(4):455–463, 2007. Cited on pp. 65 and 77.

[CKL+09] Jianer Chen, Joachim Kneis, Songjian Lu, Daniel Mölle, Stefan Richter, Peter Rossmanith, Sing-Hoi Sze, and Fenghui Zhang. Randomized divide-and-conquer: Improved path, matching, and packing algorithms. *SIAM Journal on Computing*, 38(6):2526–2547, 2009. Cited on pp. 22, 112, 131, and 179.

[CKX06] Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved parameterized upper bounds for Vertex Cover. In *Proceedings of the 31st International Symposium on Mathematical Foundations of Computer Science (MFCS '06)*, volume 4162 of *Lecture Notes in Computer Science*, pages 238–249. Springer, 2006. Cited on pp. 4 and 21.

[CLL+08] Jianer Chen, Yang Liu, Songjian Lu, Barry O'Sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *Journal of the ACM*, 55(5), 2008. Article 21, 19 pages. Cited on pp. 14, 22, and 84.

[CLS+05] Elissa J. Chesler, Lu Lu, Siming Shou, Yanhua Qu, Jing Gu, Jintao Wang, Hui C. Hsu, John D. Mountz, Nicole E. Baldwin, Michael A. Langston, David W. Threadgill, Kenneth F. Manly, and Robert W. Williams. Complex trait analysis of gene expression uncovers polygenic and pleiotropic networks that modulate nervous system function. *Nature Genetics*, 37(3):233–242, 2005. Cited on pp. 1, 2, 26, 156, 158, 174, and 179.

[CM87] Stephen A. Cook and Pierre McKenzie. Problems complete for deterministic logarithmic space. *Journal of Algorithms*, 8(3):385–394, 1987. Cited on p. 66.

[CM08] Jianer Chen and Jie Meng. On parameterized intractability: Hardness and completeness. *The Computer Journal*, 51(1):39–59, 2008. Cited on p. 17.

[Cou90] Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990. Cited on pp. 141 and 150.

[CP09] Domingos M. Cardoso and Sofia J. Pinheiro. Spectral upper bounds on the size of k-regular induced subgraphs. *Electronic Notes in Discrete Mathematics*, 32:3–10, 2009. DIMAP Workshop on Algorithmic Graph Theory. Cited on p. 65.

[CR02] Alberto Caprara and Romeo Rizzi. Packing triangles in bounded degree graphs. *Information Processing Letters*, 84(4):175–180, 2002. Cited on pp. 3, 111, and 112.

[CST03] Kathie Cameron, R. Sritharan, and Yingwen Tang. Finding a maximum induced matching in weakly chordal graphs. *Discrete Mathematics*, 266(1–3):133–142, 2003. Cited on p. 136.

[CST+07] Victoria J. Cook, Sumi J. Sun, Jane Tapia, Stephen Q. Muth, D. Fermín Argüello, Bryan L. Lewis, Richard B. Rothenberg, Peter D. McElroy, and the Network Analysis Project Team. Transmission network analysis in tuberculosis contact investigations. *Journal of Infectious Diseases*, 196:1517–1527, 2007. Cited on p. 26.

[CTW09] Zhi-Zhong Chen, Ruka Tanahashi, and Lusheng Wang. An improved randomized approximation algorithm for maximum triangle packing. *Discrete Applied Mathematics*, 157(7):1640–1646, 2009. Cited on p. 23.

[CW05] Kathie Cameron and Tracy Walker. The graphs with maximum induced matching and maximum matching the same size. *Discrete Mathematics*, 299(1–3):49–55, 2005. Cited on p. 136.

[CWC+96] Yong-Qing Cheng, Victor Wu, Robert T. Collins, Allen R. Hanson, and Edward M. Riseman. Maximum-weight bipartite matching technique and its application in image feature matching. In *Proceedings of the '96 SPIE Conference on Visual Communication and Image Processing*, volume 2727 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, pages 453–462, 1996. Cited on p. 2.

[DF95] Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness II: On completeness for W[1]. *Theoretical Computer Science*, 141(1&2):109–131, 1995. Cited on p. 79.

[DF98] Rodney G. Downey and Michael R. Fellows. Threshold dominating sets and an improved characterization of $W[2]$. *Theoretical Computer Science*, 209(1-2):123–140, 1998. Cited on pp. 17 and 59.

[DF99] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer, 1999. Cited on pp. 4, 9, 10, 16, 30, and 60.

[DFL+07] Frank K. H. A. Dehne, Michael R. Fellows, Michael A. Langston, Frances A. Rosamond, and Kim Stevens. An $O(2^{O(k)}n^3)$ FPT algorithm for the undirected feedback vertex set problem. *Theory of Computing Systems*, 41(3):479–492, 2007. Cited on pp. 14, 20, 84, 89, 90, and 91.

[DFR00] Rodney G. Downey, Michael R. Fellows, and Venkatesh Raman. The complexity of irredundant sets parameterized by size. *Discrete Applied Mathematics*, 100(3):155–167, 2000. Cited on p. 139.

[DGH+09] 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, 2009. Cited on pp. 22, 84, and 127.

[Die05] Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, 3rd edition, 2005. Cited on p. 9.

[DIM95] DIMACS. Maximum clique, graph coloring, and satisfiability. Second DIMACS implementation challenge. `http://dimacs.rutgers.edu/Challenges/`, 1995. Accessed November 2008. Cited on pp. 1, 65, 155, 166, and 170.

[Din06] Yefim Dinitz. Dinitz' algorithm: The original version and Even's version. In *Theoretical Computer Science, Essays in Memory of Shimon Even*, volume 3895 of *Lecture Notes in Computer Science*, pages 218–240. Springer, 2006. Cited on p. 42.

[DLL+06] Frank K. H. A. Dehne, Michael A. Langston, Xuemei Luo, Sylvain Pitre, Peter Shaw, and Yun Zhang. The cluster editing problem: Implementations and experiments. In *Proceedings of the 2nd International Workshop on Parameterized and Exact Computation (IWPEC '06)*, volume 4169 of *Lecture Notes in Computer Science*, pages 13–24. Springer, 2006. Cited on pp. 12 and 13.

[DLS09] 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 p. 113.

[DMZ05] William Duckworth, David Manlove, and Michele Zito. On the approximability of the maximum induced matching problem. *Journal of Discrete Algorithms*, 3(1):79–91, 2005. Cited on pp. 135 and 136.

[DS05] Irit Dinur and Shmuel Safra. On the hardness of approximating Minimum Vertex Cover. *Annals of Mathematics*, 162(1):439–485, 2005. Cited on pp. 4, 19, and 27.

[DT06] Josep Díaz and Dimitrios M. Thilikos. Fast FPT-algorithms for cleaning grids. In *Proceedings of the 23rd International Symposium on Theoretical Aspects of Computer Science (STACS '06)*, volume 3884 of *Lecture Notes in Computer Science*, pages 361–371. Springer, 2006. Cited on p. 66.

[ECFLR05] Vladimir Estivill-Castro, Michael R. Fellows, Michael A. Langston, and Frances A. Rosamond. FPT is P-time extremal structure I. In *Proceedings of the 1st Algorithms and Complexity in Durham (ACiD '05) Workshop*, volume 4 of *Texts in Algorithmics*, pages 1–41. College Publications, 2005. Cited on p. 32.

[Edm65] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965. Cited on p. 3.

[Fer05] Henning Fernau. *Parameterized Algorithmics: A Graph-Theoretic Approach*. Habilitationsschrift, Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, 2005. Cited on p. 85.

[FFR97] Ralph J. Faudree, Evelyne Flandrin, and Zdenek Ryjácek. Claw-free graphs—a survey. *Discrete Mathematics*, 164(1–3):87–147, 1997. Cited on p. 140.

[FG06] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer, 2006. Cited on pp. 4, 10, 17, 30, and 127.

[FGK+08] Fedor V. Fomin, Serge Gaspers, Dieter Kratsch, Mathieu Liedloff, and Saket Saurabh. Iterative compression and exact algorithms. In *Proceedings of the 33rd International Symposium on Mathematical Foundations of Computer Science (MFCS '08)*, volume 5162 of *Lecture Notes in Computer Science*, pages 335–346. Springer, 2008. Cited on p. 86.

[FGMN09a] Michael R. Fellows, Jiong Guo, Hannes Moser, and Rolf Niedermeier. A complexity dichotomy for finding disjoint solutions of vertex deletion problems. In *Proceedings of the 34th International Symposium on Mathematical Foundations of Computer Science (MFCS '09)*, volume 5734 of *Lecture Notes in Computer Science*, pages 319–330. Springer, 2009. Cited on pp. ix and 211.

[FGMN09b] Michael R. Fellows, Jiong Guo, Hannes Moser, and Rolf Niedermeier. A generalization of Nemhauser and Trotter's local optimization theorem. In *Proceedings of the 26th International Symposium on Theoretical Aspects of Computer Science (STACS '09)*, pages 409–420. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2009. Cited on pp. viii and 211.

[FGST89] Ralph J. Faudree, András Gyárfás, Richard H. Schelp, and Zsolt Tuza. Induced matchings in bipartite graphs. *Discrete Mathematics*, 78(1-2):83–87, 1989. Cited on p. 136.

[FHR+04] Michael R. Fellows, Pinar Heggernes, Frances A. Rosamond, Christian Sloper, and Jan Arne Telle. Finding k disjoint triangles in an arbitrary graph. In *Proceedings of the 30th International Workshop on Graph-Theoretic Conecpts in Computer Science (WG '04)*, volume 3353 of *Lecture Notes in Computer Science*, pages 235–244. Springer, 2004. Cited on pp. 112, 113, 114, 115, 116, and 178.

[FKH04] Ariel Felner, Richard E. Korf, and Sarit Hanan. Additive pattern database heuristics. *Journal of Artificial Intelligence Research*, 21:1–39, 2004. Cited on p. 85.

[FKN+07] Michael R. Fellows, Christian Knauer, Naomi Nishimura, Prabhakar Ragde, Frances A. Rosamond, Ulrike Stege, Dimitrios M. Thilikos, and Sue Whitesides. Faster fixed-parameter tractable algorithms for matching and packing problems. *Algorithmica*, 52(2):167–176, 2007. Cited on pp. 113, 121, 130, and 131.

[FLG00] Gary W. Flake, Steve Lawrence, and C. Lee Giles. Efficient identification of web communities. In *Proceedings of the sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '00)*, pages 150–160. ACM Press, 2000. Cited on p. 2.

[FLLW09] Qilong Feng, Yang Liu, Songjian Lu, and Jianxin Wang. Improved deterministic algorithms for weighted matching and packing problems. In *Proceedings of the 5th Annual Conference on Theory and Applications of Models of Computation (TAMC '08)*, volume 5532 of *Lecture Notes in Computer Science*, pages 211–220. Springer, 2009. Cited on pp. 22 and 112.

[FLRS07] Michael R. Fellows, Michael A. Langston, Frances A. Rosamond, and Peter Shaw. Efficient parameterized preprocessing for Cluster Editing. In *Proceedings of the 16th International Symposium on Fundamentals of Computation Theory (FCT '07)*, volume 4639 of *Lecture Notes in Computer Science*, pages 312–321. Springer, 2007. Cited on p. 12.

[FR09] Henning Fernau and Daniel Raible. A parameterized perspective on packing paths of length two. *Journal of Combinatorial Optimization*, 18(4):319–341, 2009. Cited on pp. 112 and 140.

[FS08] Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct PCPs for NP. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC '08)*, pages 133–142. ACM Press, 2008. Cited on p. 113.

[FT04] Fedor V. Fomin and Dimitrios M. Thilikos. Fast parameterized algorithms for graphs on surfaces: Linear kernel and exponential speed-up. In *Proceedings of the 31st International Colloquium on Automata, Languages, and Programming (ICALP '04)*, volume 3142 of *Lecture Notes in Computer Science*, pages 581–592. Springer, 2004. Cited on p. 141.

[Fuj98] Toshihiro Fujito. A unified approximation algorithm for node-deletion problems. *Discrete Applied Mathematics*, 86(2-3):213–231, 1998. Cited on pp. 19 and 27.

[Fuj99] Toshihiro Fujito. Approximating node-deletion problems for matroidal properties. *Journal of Algorithms*, 31(1):211–227, 1999. Cited on p. 19.

[FWY09] Rudolf Fleischer, Xi Wu, and Liwei Yuan. Experimental study of FPT algorithms for the directed feedback vertex set problem. In *Proceedings of the 17th Annual European Symposium on Algorithms (ESA '09)*, volume 5757 of *Lecture Notes in Computer Science*, pages 611–622. Springer, 2009. Cited on pp. 22 and 179.

[GGH+06] Jiong Guo, Jens Gramm, Falk Hüffner, Rolf Niedermeier, and Sebastian Wernicke. Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. *Journal of Computer and System Sciences*, 72(8):1386–1396, 2006. Cited on pp. 14, 20, 84, 89, 90, and 91.

[GGHN04] Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Automated generation of search tree algorithms for hard graph modification problems. *Algorithmica*, 39(4):321–347, 2004. Cited on p. 13.

[GGHN05] Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Graph-modeled data clustering: Exact algorithms for clique generation. *Theory of Computing Systems*, 38(4):373–392, 2005. Cited on pp. 12 and 13.

[GHK73] Don L. Greenwell, Robert L. Hemminger, and Joseph B. Klerlein. Forbidden subgraphs. In *Proceedings of the 4th Southeastern Conference on Combinatorics, Graph Theory and Computing*, pages 389–394, 1973. Cited on p. 8.

[GHM07] Jiong Guo, Falk Hüffner, and Hannes Moser. Feedback arc set in bipartite tournaments is NP-complete. *Information Processing Letters*, 102(2–3):62–65, 2007. Cited on p. 211.

[GIC07] Jerry Grossman, Patrick Ion, and Rodrigo De Castro. The Erdős number project. `http://www.oakland.edu/enp/`, 2007. Accessed January 2009. Cited on pp. 166 and 167.

[GJ79] 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 pp. 3, 77, 102, and 103.

[GJS76] Michael R. Garey, David S. Johnson, and Larry J. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976. Cited on p. 79.

[GKNU09] Jiong Guo, Christian Komusiewicz, Rolf Niedermeier, and Johannes Uhlmann. A more relaxed model for graph-based data clustering: s-plex editing. In *Proceedings of the 5th International Conference on Algorithmic Aspects in Information and Management (AAIM '09)*, volume 5564 of *Lecture Notes in Computer Science*, pages 226–239. Springer, 2009. Cited on p. 158.

[GL93] Martin C. Golumbic and Renu Laskar. Irredundancy in circular arc graphs. *Discrete Applied Mathematics*, 44(1–3):79–089, 1993. Cited on p. 136.

[GL00] Martin C. Golumbic and Moshe Lewenstein. New results on induced matchings. *Discrete Applied Mathematics*, 101(1–3):157–165, 2000. Cited on pp. 135 and 136.

[GL05] Zvi Gotthilf and Moshe Lewenstein. Tighter approximations for maximum induced matchings in regular graphs. In *Proceedings of the 3rd Workshop on Approximation and Online Algorithms (WAOA '05)*, volume 3879 of *Lecture Notes in Computer Science*, pages 270–281. Springer, 2005. Cited on p. 136.

[GMN09] Jiong Guo, Hannes Moser, and Rolf Niedermeier. Iterative compression for exactly solving NP-hard minimization problems. In *Algorithmics of Large and Complex Networks*, volume 5515 of *Lecture Notes in Computer Science*, pages 65–80. Springer, 2009. Cited on pp. ix, 14, 179, and 211.

[GN07a] Jiong Guo and Rolf Niedermeier. Invitation to data reduction and problem kernelization. *ACM SIGACT News*, 38(1):31–45, 2007. Cited on pp. 12 and 30.

[GN07b] 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 pp. 141 and 145.

[GNW09] Jiong Guo, Rolf Niedermeier, and Sebastian Wernicke. Fixed-parameter tractability results for full-degree spanning tree and its dual. *Networks*, 2009. To appear. Cited on pp. 141 and 142.

[GRC+98] Venkatesan Guruswami, C. Pandu Rangan, Maw-Shang Chang, Gerard J. Chang, and C. K. Wong. The vertex-disjoint triangles problem. In *Proceedings of the 24th International Workshop on Graph-Theoretic Concepts in Computer Science (WG '98)*, volume 1517 of *Lecture Notes in Computer Science*, pages 26–37. Springer, 1998. Cited on pp. 23 and 112.

[GRS06] Sushmita Gupta, Venkatesh Raman, and Saket Saurabh. Fast exponential algorithms for maximum r-regular induced subgraph problems. In *Proceedings of the 26th International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS '06)*, volume 4337 of *Lecture Notes in Computer Science*, pages 139–151. Springer, 2006. Cited on pp. 65 and 66.

[GT89] Harold N. Gabow and Robert E. Tarjan. Faster scaling algorithms for network problems. *SIAM Journal on Computing*, 18(5):1013–1036, 1989. Cited on p. 95.

[Guo06] Jiong Guo. *Algorithm Design Techniques for Parameterized Graph Modification Problems*. PhD thesis, Institut für Informatik, Friedrich-Schiller Universität Jena, 2006. Cited on pp. 22, 85, and 89.

[Guo09] Jiong Guo. A more effective linear kernelization for cluster editing. *Theoretical Computer Science*, 410(8-10):718–726, 2009. Cited on p. 12.

[HBN09] Falk Hüffner, Nadja Betzler, and Rolf Niedermeier. Separator-based data reduction for signed graph balancing. *Journal of Combinatorial Optimization*, 2009. Available electronically. Cited on pp. 84, 86, 178, and 179.

[HK73] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973. Cited on pp. 119 and 126.

[HKMN09a] Falk Hüffner, Christian Komusiewicz, Hannes Moser, and Rolf Niedermeier. Fixed-parameter algorithms for cluster vertex deletion. *Theory of Computing Systems*, 2009. Available electronically. Cited on pp. 21, 91, 95, 127, and 211.

[HKMN09b] Falk Hüffner, Christian Komusiewicz, Hannes Moser, and Rolf Niedermeier. Isolation concepts for clique enumeration: Comparison and computational experiments. *Theoretical Computer Science*, 410(52):5384–5397, 2009. Cited on pp. 174, 175, and 211.

[HNW08] Falk Hüffner, Rolf Niedermeier, and Sebastian Wernicke. Techniques for practical fixed-parameter algorithms. *The Computer Journal*, 51(1):7–25, 2008. Cited on pp. 10 and 178.

[Hoc82] Dorit S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal on Computing*, 11(3):555–556, 1982. Cited on p. 29.

[Hoc97] Dorit S. Hochbaum, editor. *Approximation Algorithms for NP-hard Problems*. PWS Publishing Company, 1997. Cited on p. 1.

[HR97] Refael Hassin and Shlomi Rubinstein. An approximation algorithm for maximum packing of 3-edge paths. *Information Processing Letters*, 63(2):63–67, 1997. Cited on p. 179.

[HS89] Cor A. J. Hurkens and Alexander Schrijver. On the size of systems of sets every t of which have an SDR, with an application to the worst-case ratio of heuristics for packing problems. *SIAM Journal on Discrete Mathematics*, 2(1):68–72, 1989. Cited on p. 112.

[Hüf07] Falk Hüffner. *Algorithms and Experiments for Parameterized Approaches to Hard Graph Problems*. PhD thesis, Institut für Informatik, Friedrich-Schiller-Universität Jena, 2007. Cited on pp. 4, 20, 85, 86, 131, 178, and 179.

[Hüf09] Falk Hüffner. Algorithm engineering for optimal graph bipartization. *Journal of Graph Algorithms and Applications*, 13(2):77–98, 2009. Cited on pp. 14, 20, 21, 84, 86, 90, 110, 178, and 179.

[IEE07] IEEE standard for information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements—Part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. *IEEE Std 802.11-2007 (Revision of IEEE Std 802.11-1999)*, pages C1–1184, 2007. Cited on p. 134.

[II09] Hiro Ito and Kazuo Iwama. Enumeration of isolated cliques and pseudo-cliques. *ACM Transactions on Algorithms*, 5(4):40:1–40:21, 2009. Cited on pp. 158, 174, and 175.

[Jon02] Bill Jones. Computational geometry database. `http://compgeom.cs.uiuc.edu/~jeffe/compgeom/biblios.html`, February 2002. Cited on p. 167.

[Kan94] Viggo Kann. Maximum bounded H-matching is MAX SNP-complete. *Information Processing Letters*, 49(6):309–318, 1994. Cited on pp. 22 and 112.

[KH78] David G. Kirkpatrick and Pavol Hell. On the completeness of a generalized matching problem. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOC '78)*, pages 240–245. ACM Press, 1978. Cited on pp. 22 and 112.

[KHMN09] Christian Komusiewicz, Falk Hüffner, Hannes Moser, and Rolf Niedermeier. Isolation concepts for efficiently enumerating dense subgraphs. *Theoretical Computer Science*, 410(38-40):3640–3654, 2009. Cited on pp. 26, 27, 52, 156, 158, 174, 175, and 211.

[Khu02] Samir Khuller. The Vertex Cover problem. *ACM SIGACT News*, 33(2):31–33, 2002. Cited on p. 30.

[Klo94] Ton Kloks. *Treewidth. Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994. Cited on pp. 14 and 16.

[KMPS04] V. S. Anil Kumar, Madhav V. Marathe, Srinivasan Parthasarathy, and Aravind Srinivasan. End-to-end packet-scheduling in wireless ad-hoc networks. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '04)*, pages 1021–1030. ACM/SIAM, 2004. Cited on pp. 3 and 135.

[KMPS07] V. S. Anil Kumar, Madhav V. Marathe, Srinivasan Parthasarathy, and Aravind Srinivasan. Provable algorithms for joint optimization of transport, routing and MAC layers in wireless ad hoc networks. In *Proceedings of the DIALM-POMC Joint Workshop on Foundations of Mobile Computing (DIALM-POMC '07)*, 2007. Cited on pp. 3 and 135.

[KMZ08] Adrian Kosowski, Michal Malafiejski, and Pawel Zylinski. Tighter bounds on the size of a maximum P_3-matching in a cubic graph. *Graphs and Combinatorics*, 24(5):461–468, 2008. Cited on p. 113.

[Kom07] Christian Komusiewicz. *Various Isolation Concepts for the Enumeration of Dense Subgraphs*. Diplomarbeit, Institut für Informatik, Friedrich-Schiller-Universität Jena, 2007. Cited on p. 158.

[Kos99] Arie M. C. A. Koster. *Frequency Assignment—Models and Algorithms*. PhD thesis, Universiteit Maastricht, Maastricht, The Netherlands, 1999. Cited on p. 135.

[Kou08] Ioannis Koutis. Faster algebraic algorithms for path and packing problems. In *Proceedings of the 35th International Colloquium on Automata, Languages, and Programming (ICALP '08)*, volume 5125 of *Lecture Notes in Computer Science*, pages 575–586. Springer, 2008. Cited on pp. 22 and 112.

[KPS04] Iyad A. Kanj, Michael J. Pelsmajer, and Marcus Schaefer. Parameterized algorithms for feedback vertex set. In *Proceedings of the 1st International Workshop on Parameterized and Exact Computation (IWPEC '04)*, volume 3162 of *Lecture Notes in Computer Science*, pages 235–247. Springer, 2004. Cited on p. 20.

[KPXS09] Iyad A. Kanj, Michael J. Pelsmajer, Ge Xia, and Marcus Schaefer. On the induced matching problem. *Journal of Computer and System Sciences*, 2009. To appear. Cited on pp. 136, 137, 138, 141, 142, 144, 149, and 154.

[KR02] Subhash Khot and Venkatesh Raman. Parameterized complexity of finding subgraphs with hereditary properties. *Theoretical Computer Science*, 289(2):997–1008, 2002. Cited on pp. 20 and 64.

[KR03] Daniel Kobler and Udi Rotics. Finding maximum induced matchings in subclasses of claw-free and P_5-free graphs, and in graphs with matching and induced matching of equal maximum size. *Algorithmica*, 37(4):327–346, 2003. Cited on pp. 135, 136, and 140.

[KR08] Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within $2 - \epsilon$. *Journal of Computer and System Sciences*, 74(3):335–349, 2008. Cited on p. 30.

[Kra99] Adam Krawczyk. The complexity of finding a second Hamiltonian cycle in cubic graphs. *Journal of Computer and System Sciences*, 58(3):641–647, 1999. Cited on p. 91.

[Kra09] Stefan Kratsch. Polynomial kernelizations for MIN $\mathrm{F}^{+}\Pi_1$ and MAX NP. In *Proceedings of the 26th International Symposium on Theoretical Aspects of Computer Science (STACS '09)*, pages 601–612. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2009. Cited on pp. 127 and 128.

[KS03] C. W. Ko and F. Bruce Shepherd. Bipartite domination and simultaneous matroid covers. *SIAM Journal on Discrete Mathematics*, 16(4):517–523, 2003. Cited on p. 135.

[KWKE09] Łukasz Kowalik, Tomasz Waleń, Marjaž Krnc, and Rok Erman. Improved induced matchings in sparse graphs. In *Proceedings of the 4th International Workshop on Parameterized and Exact Computation (IWPEC '09)*, volume 5917 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 2009. Cited on pp. 136, 137, and 138.

[Lan08] Michael A. Langston, 2008. Personal communication. Cited on p. 26.

[LBI+01] Giuseppe Lancia, Vineet Bafna, Sorin Istrail, Ross Lippert, and Russell Schwartz. SNPs problems, complexity, and algorithms. In *Proceedings of the 9th Annual European Symposium on Algorithms (ESA '01)*, volume 2161 of *Lecture Notes in Computer Science*, pages 182–193. Springer, 2001. Cited on p. 1.

[LMS09] Daniel Lokshtanov, Matthias Mnich, and Saket Saurabh. Linear kernel for planar connected dominating set. In *Proceedings of the 5th Annual Conference on Theory and Applications of Models of Computation (TAMC '08)*, volume 5532 of *Lecture Notes in Computer Science*, pages 281–290. Springer, 2009. Cited on p. 141.

[Lok08] Daniel Lokshtanov. Wheel-free deletion is W[2]-hard. In *Proceedings of the 3rd International Workshop on Parameterized and Exact Computation (IWPEC '08)*, volume 5018 of *Lecture Notes in Computer Science*, pages 141–147. Springer, 2008. Cited on p. 21.

[Loz02] Vadim V. Lozin. On maximum induced matchings in bipartite graphs. *Information Processing Letters*, 81(1):7–11, 2002. Cited on pp. 135 and 138.

[LP86] László Lovász and Michael D. Plummer. *Matching Theory*, volume 29 of *Annals of Discrete Mathematics*. North-Holland, 1986. Cited on p. 2.

[LP09] Zbigniew Lonc and Monika Pszczola. Edge decompositions into two kinds of graphs. *Discrete Mathematics*, 309(22):6368–6374, 2009. Cited on p. 23.

[LR03] Vadim V. Lozin and Dieter Rautenbach. Some results on graphs without long induced paths. *Information Processing Letters*, 88(4):167–171, 2003. Cited on p. 136.

[LSS09] Daniel Lokshtanov, Saket Saurabh, and Somnath Sikdar. Simpler parameterized algorithm for OCT. In *Proceedings of the 20th International Workshop on Combinatorial Algorithms (IWOCA '09)*, volume 5874 of *Lecture Notes in Computer Science*, pages 380–384. Springer, 2009. Cited on pp. 14, 20, 21, 84, and 110.

[LY80] 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 pp. 19, 23, 26, 27, 64, 91, 97, 98, and 157.

[LY93] Carsten Lund and Mihalis Yannakakis. The approximation of maximum subgraph problems. In *Proceedings of the 20st International Colloquium on Automata, Languages, and Programming (ICALP '93)*, volume 700 of *Lecture Notes in Computer Science*, pages 40–51. Springer, 1993. Cited on p. 19.

[Mar09] Dániel Marx. Chordal deletion is fixed-parameter tractable. *Algorithmica*, 2009. Available electronically. Cited on pp. 21, 84, and 91.

[MH09] Benjamin McClosky and Illya V. Hicks. Combinatorial algorithms for the maximum k-plex problem. Manuscript, January 2009. Cited on pp. 156, 158, 166, 167, 168, 169, 172, and 176.

[MNS09] Hannes Moser, Rolf Niedermeier, and Manuel Sorge. Algorithms and experiments for clique relaxations—finding maximum s-plexes. In *Proceedings of the 8th International Symposium on Experimental Algorithms (SEA '09)*, volume 5526 of *Lecture Notes in Computer Science*, pages 233–244. Springer, 2009. Cited on pp. viii, x, and 211.

[Mos05] Hannes Moser. *Exact algorithms for generalizations of Vertex Cover*. Diploma thesis, Institut für Informatik, Friedrich-Schiller Universität Jena, 2005. Cited on p. 211.

[Mos09] Hannes Moser. A problem kernelization for graph packing. In *Proceedings of the 33rd Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM '07)*, volume 5404 of *Lecture Notes in Computer Science*, pages 401–412. Springer, 2009. Cited on pp. ix and 211.

[MPS04] Luke Mathieson, Elena Prieto, and Peter Shaw. Packing edge disjoint triangles: A parameterized view. In *Proceedings of the 1st International Workshop on Parameterized and Exact Computation (IWPEC '04)*, volume 3162 of *Lecture Notes in Computer Science*, pages 127–137. Springer, 2004. Cited on pp. 23 and 179.

[MRS+07a] Sounaka Mishra, Venkatesh Raman, Saket Saurabh, Somnath Sikdar, and C. R. Subramanian. The complexity of finding subgraphs whose matching number equals the vertex cover number. In *Proceedings of the 18th International Symposium on Algorithms and Computation (ISAAC '07)*, volume 4835 of *Lecture Notes in Computer Science*, pages 268–279. Springer, 2007. Cited on p. 86.

[MRS07b] Hannes Moser, Venkatesh Raman, and Somnath Sikdar. The parameterized complexity of the unique coverage problem. In *Proceedings of the 18th International Symposium on Algorithms and Computation (ISAAC '07)*, volume 4835 of *Lecture Notes in Computer Science*, pages 621–631. Springer, 2007. Cited on p. 211.

[MS07a] Dániel Marx and Ildikó Schlotter. Obtaining a planar graph by vertex deletion. In *Proceedings of the 33rd International Workshop on Graph-Theoretic Concepts in Computer Science (WG '07)*, volume 4769 of *Lecture Notes in Computer Science*, pages 292–303. Springer, 2007. Cited on pp. 20, 21, and 91.

[MS07b] Hannes Moser and Somnath Sikdar. The parameterized complexity of the induced matching problem in planar graphs. In *Proceedings of the 1st International Frontiers of Algorithmics Workshop (FAW '07)*, volume 4613 of *Lecture Notes in Computer Science*, pages 325–336. Springer, 2007. Cited on p. ix.

[MS08a] Luke Mathieson and Stefan Szeider. The parameterized complexity of regular subgraphs problems and generalizations. In *Proceedings of the 14th Computing: The Australasian Theory Symposium (CATS '08)*, volume 77 of *Conferences in Research and Practice in Information Technology*, pages 79–86. Australian Computer Society, 2008. Cited on pp. 65, 80, 144, 145, and 149.

[MS08b] Luke Mathieson and Stefan Szeider. Parameterized graph editing with chosen vertex degrees. In *Proceedings of the 2nd Annual International Conference on Combinatorial Optimization and Applications (COCOA '08)*, volume 5165 of *Lecture Notes in Computer Science*, pages 13–22. Springer, 2008. Cited on p. 65.

[MS09a] Dániel Marx and Ildikó Schlotter. Parameterized graph cleaning problems. *Discrete Applied Mathematics*, 157(15):3258–3267, 2009. Cited on p. 66.

[MS09b] Hannes Moser and Somnath Sikdar. The parameterized complexity of the induced matching problem. *Discrete Applied Mathematics*, 157(4):715–727, 2009. Cited on pp. ix, 137, and 211.

[MT05] Jérôme Monnot and Sophie Toulouse. Approximation results for the weighted P_4 partition problems. In *Proceedings of the 15th International Symposium on Fundamentals of Computation Theory (FCT '05)*, volume 3623 of *Lecture Notes in Computer Science*, pages 388–396. Springer, 2005. Cited on p. 179.

[MT06] Hannes Moser and Dimitrios M. Thilikos. Parameterized complexity of finding regular induced subgraphs. In *Proceedings of the 2nd Algorithms and Complexity in Durham (ACiD '06) Workshop*, volume 7 of *Texts in Algorithmics*, pages 107–118. College Publications, 2006. Cited on p. ix.

[MT09] Hannes Moser and Dimitrios M. Thilikos. Parameterized complexity of finding regular induced subgraphs. *Journal of Discrete Algorithms*, 7(2):181–190, 2009. Cited on pp. ix and 211.

[MW08] Gordana Manic and Yoshiko Wakabayashi. Packing triangles in low degree graphs and indifference graphs. *Discrete Mathematics*, 308(8):1455–1471, 2008. Cited on p. 112.

[Nie06] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006. Cited on pp. 4, 10, 12, and 30.

[NR00] Rolf Niedermeier and Peter Rossmanith. A general method to speed up fixed-parameter-tractable algorithms. *Information Processing Letters*, 73(3–4):125–129, 2000. Cited on pp. 12, 53, and 77.

[NR03] Rolf Niedermeier and Peter Rossmanith. An efficient fixed-parameter algorithm for 3-Hitting Set. *Journal of Discrete Algorithms*, 1(1):89–102, 2003. Cited on pp. 27 and 127.

[NRT05] Naomi Nishimura, Prabhakar Ragde, and Dimitrios M. Thilikos. Fast fixed-parameter tractable algorithms for nontrivial generalizations of Vertex Cover. *Discrete Applied Mathematics*, 152(1–3):229–245, 2005. Cited on pp. 27 and 91.

[NT75] George L. Nemhauser and Leslie E. Trotter. Vertex packings: Structural properties and algorithms. *Mathematical Programming*, 8:232–248, 1975. Cited on pp. 11, 23, 25, and 29.

[OB03] Michael Okun and Amnon Barak. A new approach for approximating node deletion problems. *Information Processing Letters*, 88(5):231–236, 2003. Cited on p. 27.

[OFGZ08] Yury Orlovich, Gerd Finke, Valery Gordon, and Igor Zverovich. Approximability results for the maximum and minimum maximal induced matching problems. *Discrete Optimization*, 5(3):584–593, 2008. Cited on pp. 22 and 135.

[Öst02] Patric R. J. Östergård. A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, 120(1-3):197–207, 2002. Cited on p. 166.

[Pap94] Christos H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and System Sciences*, 48(3):498–532, 1994. Cited on p. 91.

[PdSS09] 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 p. 12.

[Pei07] Thomas Peiselt. *An Iterative Compression Algorithm for Vertex Cover*. Studienarbeit, Institut für Informatik, Friedrich-Schiller Universität Jena, 2007. Cited on pp. 85 and 110.

[Plu07] Michael D. Plummer. Graph factors and factorization: 1985-2003: A survey. *Discrete Mathematics*, 307(7-8):791–821, 2007. Cited on p. 66.

[PS98] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, 1998. Cited on p. 3.

[PS06] Elena Prieto and Christian Sloper. Looking at the stars. *Theoretical Computer Science*, 351(3):437–445, 2006. Cited on pp. 52, 112, and 113.

[PW06] M. Pretti and M. Weigt. Sudden emergence of q-regular subgraphs in random graphs. *Europhysics Letters*, 75(1):8–14, 2006. Cited on p. 64.

[PY91] Christos H. Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43(3):425–440, 1991. Cited on pp. 19 and 27.

[Rei04] Christian M. Reidys. Distance-2-matchings of random graphs. *Annals of Combinatorics*, 8(1):93–101, 2004. Cited on p. 135.

[RO09] Igor Razgon and Barry O'Sullivan. Almost 2-SAT is fixed-parameter tractable. *Journal of Computer and System Sciences*, 75(8):435–450, 2009. Cited on pp. 14 and 85.

[Rou73] N. D. Roussopoulos. A $\max\{m, n\}$ algorithm for determining the graph H from its line graph G. *Information Processing Letters*, 2(4):108–112, 1973. Cited on p. 140.

[RS04] Neil Robertson and Paul D. Seymour. Graph minors. XX. Wagner's conjecture. *Journal of Combinatorial Theory. Series B*, 92(2):325–357, 2004. Cited on p. 20.

[RS08] Venkatesh Raman and Saket Saurabh. Short cycles make W-hard problems hard: FPT algorithms for W-hard problems in graphs with no short cycles. *Algorithmica*, 52(2):203–225, 2008. Cited on pp. 139 and 180.

[RSS02] Venkatesh Raman, Saket Saurabh, and C. R. Subramanian. Faster fixed parameter tractable algorithms for undirected feedback vertex set. In *Proceedings of the 13th International Symposium on Algorithms and Computation (ISAAC '02)*, volume 2518 of *Lecture Notes in Computer Science*, pages 241–248. Springer, 2002. Cited on p. 20.

[RSS05] Venkatesh Raman, Saket Saurabh, and C. R. Subramanian. Faster algorithms for feedback vertex set. In *Proceedings of the 2nd Brazilian Symposium on Graphs, Algorithms and Combinatorics (GRACO '05)*, volume 19 of *Electronic Notes in Discrete Mathematics*, pages 273–279. Elsevier B. V., 2005. Cited on p. 20.

[RSV04] Bruce Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Operations Research Letters*, 32(4):299–301, 2004. Cited on pp. 13, 14, 20, 21, 83, 84, 89, 90, 91, and 110.

[RWB+07] Sven Rahmann, Tobias Wittkop, Jan Baumbach, Marcel Martin, Anke Truß, and Sebastian Böcker. Exact and heuristic algorithms for weighted cluster editing. In *Proceedings of the 6th International Conference on Computational Systems Bioinformatics (CSB '07)*, volume 6 of *Computational Systems Bioinformatics Conference Series*, pages 391–401. Life Sciences Society, 2007. Cited on p. 13.

[SF78] Stephen B. Seidman and Brian L. Foster. A graph-theoretic generalization of the clique concept. *Journal of Mathematical Sociology*, 6:139–154, 1978. Cited on pp. 2, 26, 156, 157, 158, and 167.

[SGW+05] Dorry L. Segev, Sommer Z. Gentry, Daniel S. Warren, Brigitte Reeb, and Robert A. Montgomery. Kidney paired donation and optimizing the use of live donor organs. *Journal of the American Medical Association*, 293(15):1883–1890, 2005. Cited on p. 2.

[Sha02] Roded Sharan. *Graph Modification Problems and their Applications to Genomic Research*. PhD thesis, Sackler Faculty of Exact Sciences, School of Computer Science, Tel Aviv University, 2002. Cited on p. 22.

[SJ96] Laura A. Sanchis and Arun Jagota. Some experimental and theoretical results on test case generators for the maximum clique problem. *INFORMS Journal on Computing*, 8(2):103–117, 1996. Cited on pp. 166 and 170.

[SMS06] Gaurav Sharma, Ravi R. Mazumdar, and Ness B. Shroff. On the complexity of scheduling in wireless networks. In *Proceedings of the 12th Annual International Conference on Mobile Computing and Networking (MOBICOM '06)*, pages 227–238. ACM, 2006. Cited on pp. 3 and 135.

[SSM06] Gaurav Sharma, Ness B. Shroff, and Ravi R. Mazumdar. Maximum weighted matching with interference constraints. In *4th IEEE Conference on Pervasive Computing and Communications Workshops (PerCom 2006 Workshops)*, pages 70–74. IEEE Computer Society Press, 2006. Cited on pp. 3 and 135.

[Ste94] Iain A. Stewart. Deciding whether a planar graph has a cubic subgraph is NP-complete. *Discrete Mathematics*, 126(1-3):349–357, 1994. Cited on p. 66.

[Ste96] Iain A. Stewart. Finding regular subgraphs in both arbitrary and planar graphs. *Discrete Applied Mathematics*, 68(3):223–235, 1996. Cited on p. 66.

[Ste97] Iain A. Stewart. On locating cubic subgraphs in bounded-degree connected bipartite graphs. *Discrete Mathematics*, 163(1-3):319–324, 1997. Cited on p. 66.

[Ste08] Iain A. Stewart. On the fixed-parameter tractability of parameterized model-checking problems. *Information Processing Letters*, 106(1):33–36, 2008. Cited on p. 65.

[STKA07] Hiroo Saito, Masashi Toyoda, Masaru Kitsuregawa, and Kazuyuki Aihara. A large-scale study of link spam detection by graph algorithms. In *Proceedings of the 3rd International Workshop on Adversarial Information Retrieval on the Web (AIRWeb '07)*, volume 215 of *ACM International Conference Proceeding Series*, pages 45–48. ACM Press, 2007. Cited on p. 2.

[SV82] Larry J. Stockmeyer and Vijay V. Vazirani. NP-completeness of some generalizations of the maximum matching problem. *Information Processing Letters*, 15(1):14–19, 1982. Cited on pp. 22, 133, and 135.

[Tho09] Stéphan Thomassé. A quadratic kernel for feedback vertex set. *ACM Transactions on Algorithms*, 2009. To appear. Cited on pp. 12, 20, 21, and 179.

[Vaz01] Vijay V. Vazirani. *Approximation Algorithms*. Springer, 2001. Cited on pp. 3 and 4.

[vBMN10] René van Bevern, Hannes Moser, and Rolf Niedermeier. Kernelization through tidying—a case study based on s-plex cluster vertex deletion. In *Proceedings of the 9th Latin American Symposium on Theoretical Informatics (LATIN '10)*, Lecture Notes in Computer Science. Springer, 2010. To appear. Cited on p. 211.

[Wah07] Magnus Wahlström. *Algorithms, Measures and Upper Bounds for Satisfiability and Related Problems*. PhD thesis, Department of Computer and Information Science, Linköpings universitet, Sweden, 2007. Cited on pp. 27 and 85.

[Wes01] Douglas B. West. *Introduction to Graph Theory*. Prentice Hall, 2nd edition, 2001. Cited on p. 9.

[WMFH04] Yuhang Wang, Fillia Makedon, James Ford, and Heng Huang. A bipartite graph matching framework for finding correspondences between structural elements in two proteins. In *Proceedings of the 26th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (IEMBS '04)*, volume 2, pages 2972–2975, 2004. Cited on p. 2.

[WNFC08] Jianxin Wang, Dan Ning, Qilong Feng, and Jianer Chen. An improved parameterized algorithm for a generalized matching problem. In *Proceedings of the 5th Annual Conference on Theory and Applications of Models of Computation (TAMC '08)*, volume 4978 of *Lecture Notes in Computer Science*, pages 212–222. Springer, 2008. Cited on pp. 52 and 112.

[WP07] Bin Wu and Xin Pei. A parallel algorithm for enumerating all the maximal k-plexes. In *Emerging Technologies in Knowledge Discovery and Data Mining*, volume 4819 of *Lecture Notes in Artificial Intelligence*, pages 476–483. Springer, 2007. Cited on pp. 156 and 158.

[Yan81a] Mihalis Yannakakis. Edge-deletion problems. *SIAM Journal on Computing*, 10(2):297–309, 1981. Cited on pp. 27 and 110.

[Yan81b] Mihalis Yannakakis. Node-deletion problems on bipartite graphs. *SIAM Journal on Computing*, 10(2):310–327, 1981. Cited on p. 19.

[YPTG06] Haiyuan Yu, Alberto Paccanaro, Valery Trifonov, and Mark Gerstein. Predicting interactions in protein networks by completing defective cliques. *Bioinformatics*, 22(7):823–829, 2006. Cited on p. 158.

[Yus07] Raphael Yuster. Combinatorial and computational aspects of graph packing and graph decomposition. *Computer Science Review*, 1(1):12–26, 2007. Cited on pp. 3, 22, 23, 111, and 112.

[Zit99] Michele Zito. Induced matchings in regular graphs and trees. In *Proceedings of the 25th International Workshop on Graph-Theoretic Concepts in Computer Science (WG '99)*, Lecture Notes in Computer Science, pages 89–100. Springer, 1999. Cited on pp. 136, 137, and 149.

Ehrenwörtliche Erklärung

Hiermit erkläre ich,

- dass mir die Promotionsordnung der Fakultät bekannt ist;
- dass ich die Dissertation selbst angefertigt und alle von mir benutzten Hilfsmittel, persönliche Mitteilungen sowie Quellen in meiner Arbeit angegeben habe;
- dass ich die Hilfe eines Promotionsberaters nicht in Anspruch genommen habe und dass Dritte weder unmittelbar noch mittelbar geldwerte Leistungen von mir für Arbeiten erhalten haben, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen;
- dass ich die Dissertation noch nicht als Prüfungsarbeit für eine staatliche oder andere wissenschaftliche Prüfung eingereicht habe;
- dass ich weder die gleiche, eine in wesentlichen Teilen ähnliche, noch eine andere Abhandlung bereits bei einer anderen Hochschule als Dissertation eingereicht habe.

Jena, Januar 2010 *Hannes Moser*

Lebenslauf

Hannes Moser
Geboren am 18. November 1978 in München

2006–	**Doktorand** Friedrich-Schiller-Universität Jena Lehrstuhl Theoretische Informatik I/Komplexitätstheorie, Prof. Dr. Rolf Niedermeier
2005	**Diplom** Diplomarbeitsthema: „Exact Algorithms for Generalizations of Vertex Cover“ [Mos05] Betreuer: Prof. Dr. Rolf Niedermeier
1999–2005	**Diplom-Studiengang Informatik**, Nebenfach theoretische Physik Friedrich-Schiller-Universität Jena
1998–1999	**Zivildienst** Bund für Umwelt- und Naturschutz Deutschland, Zweigstelle Tuttlingen
1989–1998	**Gymnasium** Gymnasium Trossingen Leistungsfächer: Mathematik und Physik
1985–1988	**Grundschule** Grundschule Maisach

Begutachtete Publikationen (bei Veröffentlichungen mit Konferenz- und Zeitschriftenversion ist nur die Zeitschriftenversion aufgeführt):

[BKM09, FGMN09b, FGMN09a, GHM07, GMN09, HKMN09a, HKMN09b, KHMN09, MNS09, Mos09, MRS07b, MS09b, MT09, vBMN10]

Jena, Januar 2010 *Hannes Moser*