Exact Algorithms for NP-hard Problems on Networks: Design, Analysis, and Implementation

Dissertation

der Fakultät für
Informations- und Kognitionswissenschaften
(Wilhelm-Schickard Institut für Informatik)
zur Erlangung des Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.)

vorgelegt von

Dipl.-Math. Jochen Alber

aus Ebersbach an der Fils

Tübingen
2002

Tag der mündlichen Qualifikation:	08.01.2003
Dekan:	Prof. Dr. Ulrich Güntzer
1. Gutachter:	Prof. Dr. Klaus-Jörn Lange
2. Gutachter:	Prof. Dr. Michael Kaufmann

Erklärung

Hiermit erkläre ich, dass ich die Arbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass alle Stellen, die im Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angaben der Quellen als Entlehnung kenntlich gemacht worden sind.

Tübingen, Oktober 2002 *Jochen Alber*

Zusammenfassung der Arbeit

Die Arbeit befasst sich mit dem Entwurf von exakten Algorithmen für NP-vollständige Optimierungsprobleme auf Graphen, wie etwa Vertex Cover, Independent Set, oder Dominating Set. Viele praxisbezogene Aufgaben, beispielsweise sogenannte "facility location"-Probleme aus dem Bereich der Entscheidungsanalyse (decision analysis), sind durch entsprechende Netzwerk-Modellierung auf derartige Fragestellungen zurückzuführen. Im Vordergrund der Arbeit stehen Lösungsverfahren mit beweisbaren Laufzeitschranken. Wegen der solchen Problemen inhärenten kombinatorischen Komplexität müssen wir exponentielles Laufzeitverhalten unserer Algorithmen in Kauf nehmen, wollen dieses jedoch kleinstmöglich halten.

Wir verfolgen dabei den Ansatz sogenannter "parametrisierter Algorithmen". Vereinfacht gesagt handelt es sich hierbei um eine zweidimensionale Herangehensweise, bei welcher die Laufzeit nicht ausschließlich in der Größe der Eingabeinstanz, sondern überdies auch in der Größe eines sogenannten "Problemparameters" gemessen wird. Problemparameter sind in unserem Zusammenhang die Größe der zu optimierenden Lösung, wie etwa die Größe einer Knotenüberdeckungsmenge bei Vertex Cover, die Größe einer unabhängigen Knotenmenge in Independent Set, oder die Größe einer dominierende Knotenmenge im Fall von Dominating Set. Ein parametrisiertes Problem[1] heißt *"fixed-parameter tractable"*, falls sich ein Lösungsalgorithmus der Laufzeit $f(k) \cdot n^{O(1)}$ finden läßt, wobei f eine beliebige, den exponentiellen Laufzeitanteil erfassende Funktion sein darf. Hier ist n die Größe der Eingabeinstanz und k der zugehörige Problemparameter. Entsprechend heißt ein solcher Algorithmus dann Fest-Parameter-Algorithmus. FPT bezeichnet die Klasse aller parametrisierten Probleme, welche fixed-parameter tractable sind.

Es ist beispielsweise bekannt, dass par-Vertex Cover in FPT liegt. Das Problem par-Independent Set hingegen ist vollständig für W[1], eine Klasse parameterisierter Probleme, die sich wohl nicht mit einem Fest-Parameter-Algorithmus lösen lassen; par-Dominating Set ist sogar W[2]-vollständig, in diesem Sinn also parametrisiert noch weniger handhabbar. Schränkt man sich auf planare Graphen ein, so behalten die genannten Probleme zwar ihre NP-Härte, sind jedoch alle in FPT.

Der Entwurf von effizienten Fest-Parameter-Algorithmen für parametrisierte Optimierungsprobleme auf *planaren* Graphen steht im Mittelpunkt der Arbeit. Unter effizient verstehen wir hierbei Algorithmen mit möglichst geringem exponentiellen Laufzeitanteil (gemessen durch die Funktion f in der Definition von FPT). Dabei untersuchen wir sowohl von theoretischer, als auch von praktischer Seite unterschiedliche Methoden des Algorithmen-Designs: Datenreduktion, beschränkte Suchbäume, Separation von Graphen und das Konzept von Baumzerle-

[1] Die "kanonische" parametrisierte Variante eines Optimierungsproblems $\mathcal{Q}$, bei welcher der Problemparameter eine Schranke auf die Optimierungskosten ist, bezeichnen wir mit par-$\mathcal{Q}$.

gungen. Jedem der Verfahren ist ein Kapitel der Arbeit gewidmet. Wir legen dabei Wert auf eine möglichst methodische, wenig problemspezifisch Herangehensweise. Als laufende Beispiele dienen u.a. die oben genannten Probleme par-Vertex Cover, par-Independent Set und par-Dominating Set.

Kapitel 1 stellt nach einer kurzen Einführung die in der Arbeit behandelten Probleme im einzelnen und die bisher bekannten Ergebnisse hierzu überblicksartig zusammen. Darüber hinaus findet sich dort ein umfangreicher Überblick über die neuen Ergebnisse dieser Arbeit.

Kapitel 2 beschäftigt sich mit sogenannter *Datenreduktion*. Dabei wird eine Eingabeinstanz in eine — bezüglich der Problemstellung äquivalente — neue Instanz kleinerer Größe, den sogenannten Problemkern, transformiert. Zentrale Forderung beim Design parametrisierter Algorithmen ist hierbei, dass die Größe des Problemkerns nicht von der Größe der Ausgangsinstanz, sondern nur noch vom Problemparameter abhängt. Wir rekapitulieren ein Theorem von Nemhauser und Trotter, welches einen sogenannten linearen Problemkern für par-Vertex Cover liefert. Außerdem zeigen wir, wie aus dem Vierfärbbarkeitssatz auf einfache Weise ein linearer Problemkern für par-Independent Set auf planaren Graphen folgt. Hauptresultat des Kapitels ist jedoch der Nachweis eines linearen Problemkerns für par-Dominating Set auf planaren Graphen, womit wir eine seit längerem offene Frage positiv beantworten können. Die Problemkernreduktion hierfür basiert auf zwei sehr einfachen und leicht zu implementierenden Reduktionsregeln; der Beweis für die Linearität des Problemkerns hierzu ist jedoch recht aufwändig.

In Kapitel 3 verfolgen wir die wohl am häufigsten angewandten Methode im Entwurf von Fest-Parameter-Algorithmen, sogenannte *beschränkte Suchbäume*. Mit deren Hilfe läßt sich eine systematische, erschöpfende Suche realisieren. Wir betrachten einen speziellen Ansatz für kombinatorische Graphprobleme, die sogenannte "degree-branching"-Technik, bei welcher ein Suchbaum sukzessive aufgrund lokaler Nachbarschaftsstrukturen im zu untersuchenden Graphen aufgebaut wird. Als einfache Beispiele dienen par-Vertex Cover und par-Independent Set auf planaren Graphen. Die Anwendung dieser Methode für par-Dominating Set auf planaren Graphen liefert einen Laufzeit $O(8^k n)$ Algorithmus, erfordert jedoch größeren Aufwand. Insbesondere die Analyse des zugehörigen Suchbaums ist sehr technisch und umfangreich. Nachdem ein früheres $O(11^k n)$ Resultat von Downey und Fellows [82, Theorem 3.1] einen (offensichtlichen) Fehler enthält, liefert unser Ergebnis — unseres Wissens nach — den ersten (korrekten) Algorithmus der Laufzeit $O(c^k n)$ für par-Dominating Set auf planaren Graphen.

In Kapitel 4 entwickeln wir Algorithmen, die eine "divide-and-conquer"-Strategie basierend auf der *Separation von Graphen* verwenden. Um eine formale, mathematisch korrekte Beschreibung dieser Methode zu ermöglichen, prägen wir den Begriff der "glueability", welcher jene Probleme charakterisiert, die sich mittels "divide-and-conquer" lösen lassen. Das Kapitel untergliedert sich in zwei Teile. Im ersten Teil beschäftigen wir uns mit Graphklassen, für welche ein sogenanntes $\sqrt{\cdot}$-Separator-Theorem bekannt ist. Wir stellen eine Methode vor, mit welcher man für parametrisiserte Probleme mit "glueability"-Eigenschaft unter gewissen Voraussetzungen Fest-Parameter-Algorithmen der Laufzeit $2^{O(\sqrt{k})} + n^{O(1)}$ gewinnen kann. Hierunter fallen, u.a., par-Vertex Cover, par-Independent Set und par-Dominating Set auf planaren Graphen. Unserer Erkenntnis nach sind dies die ersten Fest-Parameter-Algorithmen "sublinear-exponentieller" Laufzeit (d.h., die Funktion f hat einen sublinearen Exponenten) überhaupt in der Literatur. Darüber hinaus zeigen wir, dass diese Algorithmen asymptotisch optimal sind: Genauer gesagt würde ein Algorithmus der Laufzeit $2^{o(\sqrt{k})} n^{O(1)}$ für eines dieser Probleme bereits 3 SAT $\in$ DTIME($2^{o(n)}$) implizieren (n ist hierbei die Anzahl der Variablen einer 3 SAT Formel), was in der (klassischen) Komplexitätstheorie als eher unwahrscheinlich angesehen wird.

Im zweiten Teil versuchen wir die "divide-and-conquer"-Strategie auf sogenannte Kreisgraphen (disk graphs) — das sind Überschneidungsgraphen von Kreisscheiben—zu erweitern. Für diese kann jedoch kein $\sqrt{\cdot}$-Separator-Theorem im obigen Sinne gelten, da solche Graphen Cliquen unbeschränkter Größe enthalten können. Wir umgehen diese Schwierigkeit indem wir, als zentrales Ergebnis, eine neue *geometrische* Variante eines solchen Separator-Theorems für Kreisgraphen mit beschränktem Radiusverhältnis beweisen. Als Anwendung betrachten wir das par-INDEPENDENT SET Problem, welches in der Modellierung von Konfliktgraphen, u.a. etwa bei Interferenzvermeidung in Frequenzzuweisungsproblemen, auftritt. Wir geben einen Algorithmus der Laufzeit $2^{O(\sqrt{k}\log(n))}$ für Kreisgraphen mit beschränktem Radiusverhältnis, eine Laufzeit, die auf generellen Graphen — unter der Annahme 3 SAT $\notin$ DTIME($2^{o(n)}$) (n die Anzahl der Variablen) — nicht erzielt werden kann. Unter der zusätzlichen Voraussetzung der sogenannten ϑ-Präzision — dies bedeutet, dass die Mittelpunkte der Kreisscheiben gegenseitig jeweils Abstand mindestens $\vartheta > 0$ haben — finden wir für par-INDEPENDENT SET sogar einen "sublinear-exponentiellen" Fest-Parameter-Algorithmus der Laufzeit $O(2^{O(\sqrt{k})} + n)$.

In Kapitel 5 wenden wir uns dem Konzept von *Baumzerlegungen* zu. Baumzerlegungsbasierte Algorithmen setzen sich typischerweise aus zwei Phasen zusammen. In einer ersten, im allgemeinen Fall sehr rechenzeitaufwändigen Stufe wird eine Baumzerlegung (möglichst kleiner Weite) des Eingabegraphen konstruiert. Mit Hilfe dieser kann in einer zweiten Stufe das gegebene Problem in der Regel durch einen dynamischen Programmieransatz gelöst werden. Da die Weite der Baumzerlegung meist exponentiell in die Laufzeit der zweiten Phase eingeht, sind kleine Baumweiten wünschenswert. Die Arbeit liefert neue Ergebnisse bezüglich beiden Phasen der oben genannten Methode. Zum einen beweisen wir neue, konstruktive obere Schranken für die Baumweite tw(G) eines planaren Graph G. Fundamental hierbei ist der von uns eingeführte Begriff der "Layerwise Separation Property" (LSP). Es wird gezeigt, dass für jede "Ja"-Instanz (G, k) eines parametrisierte LSP-Problems eine Baumzerlegung der Weite höchstens $c\sqrt{k}$ effizient in Zeit $O(\sqrt{k}n)$ konstruiert werden kann. Speziell folgen hieraus für die Baumweite tw(G) eines planaren Graph G etwa die Schranken

$$\mathrm{tw}(G) \le 4\sqrt{3\,\mathrm{vc}(G)} + 5 \quad \text{und} \quad \mathrm{tw}(G) \le 6\sqrt{34\,\mathrm{ds}(G)} + 8,$$

wobei vc(G) die Größe einer optimalen Knotenüberdeckung (vertex cover number) bzw. ds(G) die Größe einer optimalen dominierenden Knotenmenge (domination number) bezeichnen. Beide Schranken sind asymptotisch optimal.

Zum anderen beschäftigen wir uns mit dynamischem Programmieren auf Baumzerlegungen für eine Reihe von "Dominierungs"-Problemen. Durch ein Konzept, das auf einer Art "Monotonieverhalten" innerhalb des dynamischen Programmierens beruht, verbessern wir die hierzu bisher besten Algorithmen der Literatur erheblich. U.a. geben wir einen Algorithmus der Laufzeit $O(4^{\ell} N)$ für DOMINATING SET an (ℓ ist die Weite der gegebenen Baumzerlegung und N die Anzahl der Knoten im Baum) — der zuvor beste bekannte Algorithmus hierzu hatte Laufzeit $O(9^{\ell} N)$ [183, 184]. Ähnliche Verbesserungen erzielen wir beispielsweise für VERTEX

Cover, Independent Dominating Set, Total Dominating Set, Perfect Dominating Set, oder Perfect Code.

Die Verschmelzung der Ergebnisse zu den beiden Phasen führt zu einem universellen Schema, mit welchem, unter gewissen Voraussetzungen, Fest-Parameter-Algorithmen mit Laufzeitverhalten $O(2^{O(\sqrt{k})}n)$ oder $2^{O(\sqrt{k})}+n^{O(1)}$ (d.h. Algorithmen mit sublinearem Term im exponentiellen Laufzeitanteil) für viele parametrisierte LSP-Probleme gewonnen werden können. Insbesondere erhalten wir einen $O(2^{4\sqrt{3k}}k+kn)$ Algorithmus für par-Vertex Cover, einen $O(2^{4\sqrt{6k}}k+n^2)$ Algorithmus für par-Independent Set und einen $O(2^{12\sqrt{17\log(3)k}}k+n^3)$ Algorithmus für par-Dominating Set auf planaren Graphen. Vergleichbare Ergebnisse gelten für eine Reihe von Variationen dieser Probleme, u.a., für sogenannte par-Dominating Set with Property P Probleme oder für das par-Face Cover Problem.

In Kapitel 6 stellen wir ein Software-Paket vor, welches im Rahmen dieses Projektes mit studentischer Unterstützung entwickelt wurde und eine Vielzahl der genannten Algorithmen implementiert. Wir berichten über eine Reihe von empirischen Studien zur Auswertung der Praxistauglichkeit dieser Algorithmen. Die Leistungstests basieren auf kombinatorisch generierten Zufallsgraphinstanzen. In den Tests bestätigt sich, dass unsere theoretische Analyse der Worst-Case-Szenarien oftmals (viel) zu pessimistisch für tatsächliche Aussagen über die praktische Güte der Algorithmen ist. Im einzelnen wollen wir folgende Beobachtungen hervorheben: Die Tests zum $O(8^k n)$ Suchbaum-Algorithmus für par-dominating set ergaben eine durchschnittlichen Verzweigungsgrad im Suchbaum von 1.6, der schlechtest aufgetretene Fall überhaupt hatte einen Verzweigungsgrad von 4 — nach der Worst-Case-Abschätzung wäre jedoch ein Verzweigungsgrad von 8 möglich. Bei den Tests zu den baumzerlegungsbasierten Algorithmen ergab sich ein ähnliches Bild. Die in der Praxis erhaltenen Baumzerlegungen hatten eine bei weitem geringere Baumweite, als dies aus der theoretischen Analyse zu erwarten gewesen wäre. Folglich erweist sich das gesamte Konzept der Baumzerlegungen in unseren Tests als sehr vielversprechend und effizient. Schließlich ist die eindrucksvolle Stärke der Datenreduktions-Algorithmen hervorzuheben. Wohingegen unsere Tests ergaben, dass die von Nemhauser und Trotter vorgeschlagene Problemkernreduktion für par-Vertex Cover die Eingabeinstanzen um durchschnittlich ca. 65% verkleinerte, konnten die von uns entwickelten Datenreduktionsregeln für par-Dominating Set die Eingabegraphen gar um über 99% reduzieren. Durch Hinzunahme dieser effizienten Problemkernreduktionen als "Preprocessing" konnten demnach die meisten Algorithmen signifikant beschleunigt werden.

Am Ende der Arbeit (Kapitel 7) werden die Ergebnisse nochmals zusammengefasst. Darüber hinaus berichten wir über weitere aktuelle Veröffentlichungen im Umfeld dieser Arbeit und skizzieren einen Ausblick mit Fragestellungen für zukünftige Forschungsprojekte.

Preface

"Weitermachen!"

This thesis emerged from my efforts in the design, analysis, and implementation of parameterized algorithms. The occupation in this research field goes back to my work in the project "Parameterized Complexity and Exact Algorithms (PEAL)," NI 369/1-1,1-2, supported by the Deutsche Forschungsgemeinschaft (DFG). I owe sincere thanks to this support and to Klaus-Jörn Lange and Rolf Niedermeier who initiated this research project. In particular, I want to highlight the indefatigable efforts of Rolf Niedermeier who supervised my research in this project. I am very grateful for his support. Besides the above mentioned members of our working group, I want to thank Henning Fernau, Jens Gramm, Jiong Guo, Michael Kaufmann, and Klaus Reinhardt for various stimulating discussions and for providing a very friendly and enriching working atmosphere in Tübingen. Finally, I wish to thank Michael R. Fellows, Venkatesh Raman and Jan Arne Telle for their kind invitations to common research meetings and for their hospitality during the corresponding visits.

The most important research partners in this project for me were Rolf Niedermeier and Henning Fernau. Other people from whom I profited relevantly are Hans L. Bodlaender, Ton Kloks, Michael R. Fellows, Jiří Fiala, and several other people with whom I had collaborations and discussions on the topic of this work. This thesis is based on my various research collaborations together with one or another of the above mentioned authors. In this work, however, I only present results to which I contributed in a significant manner and to whose achievement I played a major role.

In the following, I give a detailed, chapterwise description of my main personal contributions to the new results presented here.

The work is organized in six chapters followed by a short conclusion. The contents of these chapters is worked up in a fully personally biased way. After a quick "Introduction" (Chapter 1), various techniques to design fixed-parameter algorithms for combinatorial problems mainly on planar graphs are presented, namely "Data Reduction" (Chapter 2), "Bounded Search Trees" (Chapter 3), "Graph Separation" (Chapter 4), and "Tree Decomposition Based Algorithms" (Chapter 5). Finally, I report on "Experimental Studies" (Chapter 6). Nearly all chapters are organized in a similar way. They contain a first subsection (called "Background") where the reader shall be introduced to the given chapter. Then, in each chapter further sections follow in which the new results are presented.

In Chapter 2, a linear problem kernel for par-DOMINATING SET on planar graphs (see Section 2.2) is given. Coming up with the final set of reduction rules needed to prove this result mainly goes back to my work. In particular, I developed the concept of region decompositions which can be considered to be the breakthrough in proving the linear problem kernel. Finally, using this concept, all the major parts of the technically involved proof for the linear kernel were carried out by myself.

In Chapter 3, a search tree algorithm based on degree-branching is presented for par-DOMINATING SET on planar graphs (see Section 3.2). The key difficulty there lies in proving a so-called "branching theorem" for certain reduced black and white graphs. The proof for this theorem was unclear since it may not be assumed that the black induced subgraph is connected. I came up with the final proof to overcome these difficulties which finally led to the complete analysis of the search tree size.

In Chapter 4, a parameterized divide-and-conquer approach based on graph separation is studied to design—seemingly for the first time—subexponential fixed-parameter algorithms (see Section 4.2). This idea was initiated by me. In particular, I provided a mathematically sound definition and analysis of the technically involved concept of "glueability" which characterizes problems for which such an approach is possible. The second part of Chapter 4 (see Section 4.3) concentrates on disk graphs for which such separator theorems do not exist. To cope with this dilemma I provide a new geometric version of a separator theorem. The idea for this goes back to me. Also, I did the main part of the proof—which includes the decisive idea of considering the so-called covering grid of a disk graph.

In Chapter 5, a prospective tree decomposition based approach is presented. Here, my contribution was to develop the technical concept of the "Layerwise Separation Property" (LSP) to obtain fixed-parameter algorithms with a sublinear exponential growth in the running time for a wide class of problems on planar graphs (see Section 5.2). Besides, it is one of my achievements to use the LSP in order to efficiently construct tree decompositions with a guaranteed upper bound on the treewidth that is sublinear in the problem parameter. This implies new worst-case upper bounds for the treewidth of a planar graph with respect to the vertex cover number or the domination number. In a second part of Chapter 5 (see Section 5.3), the concept of dynamic programming on tree decompositions is studied. Here, my achievement is to elaborate and realize the so-called monotonicity concept for dynamic programming that results in improved algorithms for various domination-like problems. Finally, Chapter 5 is concluded by putting together the results from the previous sections leading to the claimed "sublinear-exponential" fixed-parameter algorithms (see Section 5.4).

In Chapter 6, we report on first empirical studies on most of the algorithms presented in this work. Accompanying our theoretical studies, a software package implementing these algorithms had to be designed. This project was initiated, coordinated and supervised by myself. I want to thank the students Frederic Dorn (who did the main programming work), Nadja Betzler and Simone Lehnert for their contributions in this programming project.

Tübingen, October 2002 *Jochen Alber*

Contents

Chapter 1

Introduction

1.1 Fixed-Parameter Algorithms

We encounter (hard) combinatorial optimization problems almost everywhere. As a typical example consider a so-called facility location problem where the task is to place facilities under minimum costs in such a way that all clients can profit from the facilities. We might think of placing emergency stations in a transportation network such that quick access to each point in the network can be guaranteed, or locating antennas in a mobile communication network in a way that each customer can be served. A concrete model of the facility location problem can be given using a (bipartite) graph that encodes the information which client could be served by which facility. In this graph, we have a set of vertices representing possible locations for the facilities and a set of vertices representing the clients. An edge is drawn between a vertex corresponding to a location and a vertex that corresponds to a client if and only if the client would be served by a facility placed in this location. Let us call this graph the *facility location graph*. Since placing a facility is expensive the goal is to minimize the number of facilities to "dominate" all clients.

As complexity theorists we translate this into a decision problem.

The FACILITY LOCATION **problem:**
Input: A facility location graph G and a positive integer k.
Question: Can we satisfy all clients using at most k facilities?

We ask for an algorithm solving this problem, where the running time of the algorithm shall be measured in the size of the input. Unless we are dealing with a special class of input graphs, the bad news then probably is that the FACILITY LOCATION problem is NP-complete, i.e., we do not expect that the problem can be solved in polynomial time unless P = NP. Besides heuristic methods [146] which often lack theoretical analysis, or randomized algorithms [152] the main contribution of theoretical computer science on the attack of intractability so far has been to design and analyze approximation algorithms [30, 116]. However, since establishing a facility might be very expensive we are not satisfied with an approximate solution. Consequently, our goal should be to directly attack the NP-hard problem by providing a deterministic, *exact* algorithm; in this case, however, we have to deal with exponential running times.

It is an immediate question to ask: Where does the combinatorial hardness of the FACILITY LOCATION problem come from? Is the intractability a matter of the size of the input graph G or is it a matter of the size k of the solution we seek for? Classical complexity theory does not distinguish between different parts of the input. A recent proposal on a more fine-grained approach which tries to answer the above raised questions is given by "Parameterized Complexity," a theory pioneered by Downey and Fellows and some of their co-authors [82]. The goal of the theory is to get a deeper insight into the hardness of combinatorial problems. The idea of parameterized complexity study can easily be sketched as follows. Instead of considering the input as a whole a certain part of the input is extracted as a *parameter*. In our example, the following would be a natural parameterized version of the problem.

The parameterized FACILITY LOCATION problem:

Input: A facility location graph G and a positive integer k.
Parameter: The positive integer k.
Question: Can we satisfy all clients using at most k facilities?

The running time of an algorithm solving this problem shall now be measured in the size of the graph G *and* the parameter k, separately. In this sense, whereas classical complexity theory offers a *one-dimensional* approach, parameterized complexity theory is a *two-dimensional* study of combinatorial problems.

Observe that, if the graph G has n vertices, it is trivial to solve the FACILITY LOCATION problem in time $O(n^{k+2})$—simply by trying all possible locations of placing k facilities. The decisive question now is whether we can shift the combinatorial explosion *only* into k. In other words, we ask whether the seemingly unavoidable inherent combinatorial explosion of the problem can be restricted to the parameter only, i.e., whether there exists an algorithm with running time polynomial[1] in the size of the graph G and *only* exponential in k.

More formally, a *parameterized problem* is a two-dimensional language $\mathcal{L} \subseteq \Sigma^* \times \mathbb{N}$, where Σ is some alphabet. The second coordinate of an element $(x, k) \in \mathcal{L}$ is called the *parameter.*[2]

As a convention throughout this work, if not otherwise stated, we will always use n for the size $|x|$ of an input instance x.

Definition 1.1.1 *A parameterized problem $\mathcal{L}$ is called* fixed-parameter tractable *if there exists an algorithm that decides, for an instance $(x, k) \in \Sigma^* \times \mathbb{N}$, the word problem $(x, k) \overset{?}{\in} \mathcal{L}$ in time $f(k) \cdot n^{O(1)}$, where $n = |x|$ and f is an arbitrary function that only depends on k. The associated complexity containing all parameterized problems that are fixed-parameter tractable is called* FPT.

In other words, the function f captures the exponential part of the running time. For each fixed k, the problem is solvable in polynomial time where the degree of the polynomial is

[1]The degree of the polynomial shall be independent of k.

[2] In this work, the parameter will always be a number. In a more general setting, a parameter could also be a substructure encoded over some alphabet Γ.

independent of k. The hope is that for a small value of k, the "constant $f(k)$" is reasonably small, yielding an efficient exact algorithm for the problem. Specifically, in our above given example, the number of facilities we wish to place (i.e., the parameter k) might be small compared to the number of possible locations for the facilities (which are encoded in the graph G). Hence, a fixed-parameter algorithm (if existent)—e.g., of running time $O(2^k n)$—would be of practical use to compute an exact solution for the problem and, thus, to cope with its NP-hardness.

Not every parameterized problem is necessarily fixed-parameter tractable. In fact, analogously to classical complexity theory, Downey and Fellows developed a completeness program [82] for classifying parameterized problems. However, the completeness theory of parameterized intractability involves significantly more technical effort. We very briefly sketch the most relevant parts of this theory in the following (for details we refer to [82]).

In order to compare the hardness of parameterized problems, in complete analogy to the classical complexity, a (two-dimensional) notion of classical reduction—called *parameterized reduction*—is introduced (for a precise definition we refer to [82, Definition 9.3] or to Definition 4.2.8 in Section 4.2.2). The "lowest class of parameterized intractability," so-called W[1], can be defined as the class of parameterized languages for which a parameterized reduction to the so-called SHORT TURING MACHINE ACCEPTANCE problem (also known as the k-STEP HALTING problem) exists [82]. Here, we want to determine for an input consisting of a nondeterministic Turing machine M and a string x, whether M has a computation path accepting x in at most k steps. In some sense, this is the parameterized analogue to the TURING MACHINE ACCEPTANCE problem—the basic generic NP-complete problem in classical complexity theory. Downey and Fellows argue that "if one accepts the philosophical argument that TURING MACHINE ACCEPTANCE is intractable, then the same reasoning would suggest that SHORT TURING MACHINE ACCEPTANCE is fixed-parameter intractable." Moreover, the working hypothesis that FPT and W[1] do not coincide has a strong link to classical complexity study: It was shown by Abrahamson *et al.* [1] that FPT = W[1] implies that 3 SAT $\in$ DTIME($2^{o(n)}$), n being the number of variables of a boolean formula. It is open whether 3 SAT can be solved deterministically in exponential time with a sublinear exponent. However, it is generally believed not to hold true.[3] The result of Abrahamson *et al.* was complemented by Cai and Juedes [53] who showed that SAT $\in$ DTIME($2^{o(n)}$) (again n being the number of variables) has the consequence that FPT = W[1].

As a matter of fact, W[1] only is the lowest level of a whole hierarchy of parameterized intractability. In general, the classes W[t], $t \in \mathbb{N}_{\geq 1}$, are defined based on "logical depth" (i.e., the number of alternations between unbounded fan-in And- and Or-gates) in boolean circuits [82, Definition 10.2]. It must be emphasized, however, that the majority of *natural* parameterized problems seems to be in FPT, or complete for either of the classes W[1] or W[2]. In this sense, these three classes probably are the most fundamental ones in parameterized complexity study. We will see examples for these classes in the following subsection. For further details, surveys on the field of parameterized complexity and overviews on the recent developments, the reader is referred to [12, 15, 82, 83, 84, 85, 92, 155, 165].

Lately, two lines of research emerged from the concept of parameterized complexity study. One direction focuses on the classification of various parameterized problems along the W-

[3] The best known (deterministic) algorithm so far for 3 SAT has running time $O(1.481^n)$[115].

hierarchy (for an extensive list of already classified problems, see [82]). Another direction is concerned with a closer algorithmic study of the problems inside the class FPT and the design of *efficient* fixed-parameter algorithms. This latter algorithmic aspect lies at the heart of this thesis. Note that in the definition of the class FPT we allow the function f, which captures the exponential part of the running time, to be arbitrary. Typical functions that appear for fixed-parameter algorithms in the literature are, e.g., $f(k) = c^k$, $f(k) = k^k$, or $f(k) = c^{k^2}$. Still, it could be as bad as $f(k) = (\cdots((k!)!)!\cdots)!$. The special focus in the design of fixed-parameter algorithm in this thesis will lie on the (asymptotic) quality of the function f. In particular, we will present various algorithms where the function f grows sublinearly in the exponent, namely $f(k) = c^{\sqrt{k}}$ for some constant c. Hence, our studies are interesting both from an algorithmic point of view as well as from a structural point of view allowing further insight into the class FPT.

Graph theory and related computational problems so far appear to be among the most fertile grounds for the study of parameterized problems. The scope of this work is to present various tools and techniques to design fixed-parameter algorithms for optimization problems on (planar) graphs. Our focus will be on methods such as "data reduction by problem kernelization," "bounded search trees," "graph separation," and "tree decomposition based" algorithms. The aim is to evaluate the applicability and the value of the distinct methods—in first place, from a theoretical, but also from a practical point of view. Up to now, several interesting but specialized fixed-parameter algorithms have been developed. The thrust has been to improve running times in a problem-specific manner. It is a crucial goal throughout this work not to narrowly stick to problem-specific approaches, but to try to widen the techniques as far as possible.

1.2 Preliminaries and Notion

Graphs and, in particular, planar graphs will be the main object of study in this work. We assume basic familiarity with the notion of graphs and standard algorithmic tools. For a very comprehensive overview on the general theory of graphs we refer to [74]; we point to [159] for a textbook on planar graphs; and for an overview on graph classes, the reader is referred to [50]. We use the following notion throughout this work.

Graphs and (Induced) Subgraphs. Unless otherwise specified, we always deal with undirected graphs that contain no multiple edges and no self loops. A *graph* is denoted by $G = (V, E)$, where V is the set of *vertices* and $E \subseteq V \times V$ is the set of *edges*. If not otherwise stated, n will always refer to the number of vertices in a graph and m will refer to the number of edges. To stress that the vertices V (or edges E, respectively) belong to G, we sometimes use the notion $V(G)$ (or $E(G)$, respectively) instead. A *subgraph* $G' = (V', E')$ of G is a graph with $V' \subseteq V$ and $E' \subseteq E \cap (V' \times V')$. For a subset $V' \subseteq V$, the graph $G[V']$ is called the *induced* subgraph, i.e., the subgraph whose vertex set is V' and whose edge set is $E \cap (V' \times V')$.

Neighborhood and Degree. Let $v \in V$ be a vertex. Then, $N(v) := \{w \in V : \{v, w\} \in E\}$ denotes the *(open) neighborhood* of v, and $N[v] := N(v) \cup \{v\}$ denotes the *closed neighborhood*of v. For $V' \subseteq V$, we use the abbreviations $N(V') := \bigcup_{v \in V'} N(v)$ and similarly $N[V'] := \bigcup_{v \in V'} N[v]$. We write $\deg(v)$ for the *degree* of the vertex v, where $\deg(v) := |N(v)|$. To avoid ambiguity and to emphasize the underlying graph, we sometimes use $N_G(v)$, $N_G[v]$, or $\deg_G(v)$ instead.

Graph Operations. Occasionally, we need the operation of deleting or adding vertices or edges to a graph G. Suppose $v, w \in V(G)$ and $e \in E(G)$. By $G-v$ we denote the subgraph of G, where v is removed together with all incident edges, i.e., $V(G-v) := V \setminus \{v\}$ and $E(G-v) := E \setminus \{e \in E(G) : v \in e\}$. Similarly, $G - e$ defines the subgraph of G where the edge e is removed. Furthermore, $G + \{v, w\}$ is the supergraph of G where the edge $\{v, w\}$ is added (if not already existent). A graph H is called a *minor* of G if it is isomorphic to a graph G' that is obtained by repeated application of edge contractions to a subgraph of G. Here, an *edge contraction* is the operation of removing an edge $e = \{v, w\} \in E(G)$ from a given graph G and identifying the endpoints into a new vertex v_e, i.e., for the resulting graph G/e, formally, we get $V(G/e) := (V(G) \setminus e) \cup \{v_e\}$, and $E(G/e) := \{ h \in E(G) : e \cap h = \emptyset \} \cup \{ \{u, v_e\} : u \in N(v) \cup N(w) \}$.

Paths, Connected Components, Spanning Forest. A *path* between two vertices $v, w \in V(G)$ is a set of edges $e_1, \ldots, e_\ell \in E(G)$, such that $v \in e_1$, $w \in e_\ell$, and $|e_i \cap e_{i+1}| = 1$, for $1 \leq i \leq \ell - 1$, and $e_i \cap e_j = \emptyset$, for $1 \leq i, j \leq \ell$ with $|i - j| > 1$. A graph $G = (V, E)$ is called *connected* if there is a path between any pair of distinct vertices $v, w \in V(G)$. A *connected component* of a graph G is a maximal connected subgraph of G. Finally, a *spanning tree* of a connected graph G is a tree T that is a subgraph of G and uses all vertices, i.e., $V(T) = V(G)$. A *spanning forest* of a general graph G is a set of spanning trees for each connected component of G.

Planar Graphs, Graphs of Bounded Genus. In our setting, we mainly deal with planar graphs, i.e., graphs that can be embedded in the (Euclidean) plane without any edge-crossing. More formally, an *embedding* of a graph in the plane[4] is a mapping ϕ that maps each vertex $v \in V(G)$ to a point $\phi(v) \in \mathbb{R}^2$, and each edge $e \in E(G)$ to a (non-self-intersecting) polygonal arc $\phi(e) \subseteq \mathbb{R}^2$, in such a way that for each edge $\{v, w\}$ the endpoints of the arc $\phi(\{v, w\})$ are the points $\phi(v)$ and $\phi(w)$. An embedding ϕ of G is called *crossing-free*, if, for all distinct pairs $e_1, e_2 \in E$, we have $\phi(e_1) \cap \phi(e_2) = \emptyset$ in case $e_1 \cap e_2 = \emptyset$ and $\phi(e_1) \cap \phi(e_2) = \phi(v_{e_1,e_2})$ in case $e_1 \cap e_2 = \{v_{e_1,e_2}\}$. A crossing-free embedding cuts the plane in various regions that are called *faces* (of the embedding). More precisely, the faces are the topological connected components of the set $\mathbb{R}^2 \setminus \left(\bigcup_{e \in E} \phi(e)\right)$. The (only) open set among the faces is called the *exterior face*. A graph G is said to be *planar* if a crossing-free embedding ϕ in the plane exists. A well-known theorem due to Kuratowski (see, e.g., [159, Theorem 1.3]) states that a graph is planar if and only if it contains no K_5 (complete graph of five vertices) and no $K_{3,3}$ (complete bipartite graph with three vertices in each bipartition set) as a minor. A graph can be tested for planarity and, if planar, it can be embedded in the plane in linear time (see, e.g., [59, 117]). The pair (G, ϕ) is called a *plane graph*. Planar graphs enjoy many properties. In our context, Euler's formula (dating back to Euler, 1750) is the most important one. For every planar graph $G = (V, E)$, we have

$$|V| - |E| + |F| = 1 + c_G, \tag{1.1}$$

where F denotes the set of faces of G and c_G denotes the number of connected components of G [159, Theorem 1.2]. From this formula it is not hard to derive that every planar graph contains a vertex of degree at most 5 [159, Corollary 1.4].

Euler's formula even holds in the more general context of graphs that are embedded on (orientable) surfaces: Let $\mathcal{S}_g$ denote the orientable surface of genus g that is obtained by "adding

[4] The notion of an embedding can be generalized to any other surface.

g handles" to the sphere $\mathcal{S}$ (see [150, Chapter 3.1] for details), and let $\mathbb{G}(\mathcal{S}_g)$ denote the class of graphs that admit a crossing-free embedding on $\mathcal{S}_g$. The *genus* of a graph is defined by $g(G) := \min\{ g \geq 0 : G \in \mathbb{G}(\mathcal{S}_g) \}$. For fixed g, there is a linear time algorithm which tests whether a graph belongs to $\mathbb{G}(\mathcal{S}_g)$, and, if true, returns a corresponding embedding [149]. For each graph G with $g = g(G)$ and a crossing-free embedding in $\mathcal{S}_g$, the generalized Euler formula states that

$$|V| - |E| + |F| = 2 - 2g \tag{1.2}$$

(see [150, Eqn. (3.7)]). Using the easy observation that $2|E| \geq 3|F|$, a simple calculation shows

$$|E| \leq 3|V| - 6 + 6g, \tag{1.3}$$

which holds for all graphs $G \in \mathbb{G}(\mathcal{S}_g)$. In particular, this implies that a graph of bounded genus can have at most $O(|V|)$ many edges, whereas a general graph may have up to $|V|(|V| - 1)/2$ many edges.

Computational Models. The algorithms presented in this work are analyzed upon the hypothetical model of a unit-cost Random Access Machine (RAM) as introduced by Cook and Reckhow [64]. In this model each simple operation (such as arithmetic operations, compares, memory access, etc.) takes one time step. This has become the most popular and most widely used standard model for algorithm analysis since it strikes the balance by capturing the most essential behavior of computers while at the same time being simple to work with. All algorithms are analyzed using the so-called "O-notation" (see, e.g., [65, Chapter 3]).[5] The running time of a graph algorithm is always measured in the size $|G|$, i.e., the number n of vertices and m of edges. Most of the algorithms will be for planar graphs or, more generally, for graphs from $\mathbb{G}(\mathcal{S}_g)$. By the above observations, the size $|G|$ of a graph in $\mathbb{G}(\mathcal{S}_g)$ is given by $O(n)$.

1.3 Hard Optimization Problems on Graphs

In our work, we mainly deal with optimization problems. The following, very general definition is taken from [161, Definition 13.1].

Definition 1.3.1 *An optimization problem is a 4-tupel* $\mathcal{Q} = (\mathrm{I}_\mathcal{Q}, F_\mathcal{Q}, c_\mathcal{Q}, \mathrm{opt}_\mathcal{Q})$*, where*

(i) $\mathrm{I}_\mathcal{Q}$ *is a set of input instances (recognizable in polynomial time)*

(ii) $F_\mathcal{Q}(x)$ *is the set of* feasible solutions *for some* $x \in \mathrm{I}_\mathcal{Q}$*,*

(iii) $c_\mathcal{Q}(y)$ *is the* cost *for the solution* $y \in F_\mathcal{Q}(x)$ *of some* $x \in \mathrm{I}_\mathcal{Q}$*, and*

(iv) $\mathrm{opt}_\mathcal{Q} \in \{\min, \max\}$.

[5] We sometimes use an "interleaved" form of the O-notation, e.g., we use a term of the form $O(2^{O(\sqrt{k})}n)$. To stay within a mathematically precise framework, for two functions $f, g : \mathbb{N} \to \mathbb{N}$, we let $f(O(g)) := \bigcup_{g' \in O(g)} \{f \circ g'\}$, and we define $O(\mathcal{F}) := \bigcup_{f \in \mathcal{F}} O(f)$, for a family of functions $\mathcal{F}$.

For an input instance $x \in I_{\mathcal{Q}}$*, we define the* optimal cost *to be*

$$\mathrm{OPT}_{\mathcal{Q}}(x) = \mathrm{opt}_{\mathcal{Q}}\{\, c_{\mathcal{Q}}(z) \mid z \in F_{\mathcal{Q}}(x)\,\}.$$

An optimal solution *for an input instance* $x \in I_{\mathcal{Q}}$ *is a feasible solution* $y \in F_{\mathcal{Q}}(x)$ *which has optimal cost, i.e.,* $c_{\mathcal{Q}}(y) = \mathrm{OPT}_{\mathcal{Q}}(x)$*.*

An optimization problem induces, in a very natural way, a parameterized problem, simply by parameterizing on the target cost function (see [82, Section 4.1]).

Definition 1.3.2 *Let* $\mathcal{Q} = (I_{\mathcal{Q}}, F_{\mathcal{Q}}, c_{\mathcal{Q}}, \mathrm{opt}_{\mathcal{Q}})$ *be an optimization problem. The* parameterized problem associated with $\mathcal{Q}$ *which will be denoted by* par-$\mathcal{Q}$ *is given as follows.*

The parameterized problem par-$\mathcal{Q}$**:**

Input: *An instance* $x \in I_{\mathcal{Q}}$ *and a positive integer* k*.*
Parameter: *The positive integer* k*.*
Question: *Does* $\left\{ \begin{array}{ll} \mathrm{OPT}_{\mathcal{Q}}(x) \geq k, & \text{if } \mathrm{opt}_{\mathcal{Q}} = \max, \\ \mathrm{OPT}_{\mathcal{Q}}(x) \leq k, & \text{if } \mathrm{opt}_{\mathcal{Q}} = \min, \end{array} \right\}$ *hold?*

This decision problem corresponds to the parameterized language

$$\begin{array}{lll} \text{par-}\mathcal{Q} := \{\,(x,k) \mid x \in I_{\mathcal{Q}}, \mathrm{OPT}_{\mathcal{Q}}(x) \geq k\,\}, & \text{if } \mathrm{opt}_{\mathcal{Q}} = \max, & \text{or} \\ \text{par-}\mathcal{Q} := \{\,(x,k) \mid x \in I_{\mathcal{Q}}, \mathrm{OPT}_{\mathcal{Q}}(x) \leq k\,\}. & \text{if } \mathrm{opt}_{\mathcal{Q}} = \min. & \end{array}$$

We say that we solve *the parameterized problem* par-$\mathcal{Q}$ constructively*, if, in case of giving a positive answer to the decision question, we furthermore output a feasible solution* $y \in F_{\mathcal{Q}}(x)$ *that witnesses* $(x, k) \in$ par-$\mathcal{Q}$*.*[6]

Throughout this work, the running time of an algorithm for a parameterized problem par-$\mathcal{Q}$ will be measured in the size $n = |x|$ and the parameter k.

In the remainder of this section, we introduce the three optimization problems

VERTEX COVER, INDEPENDENT SET, and DOMINATING SET.

These three problems will serve as our running examples throughout this work. There are two main reasons, why this "threesome" of problems stirs our special attention:

- The problems are probably (among) the most important optimization problems in combinatorial graph theory. To underline this statement we quote from [110, Preface]:

  "[...] perhaps the fastest-growing area within graph theory is the study of domination and related subset problems, such as independence, covering, and matching [...]"

[6] Unless otherwise stated, all algorithms in this work will solve the parameterized problems constructively. Hence, as a convention, saying that "an algorithm solves the parameterized problem" will implicitly refer to a constructive solution.

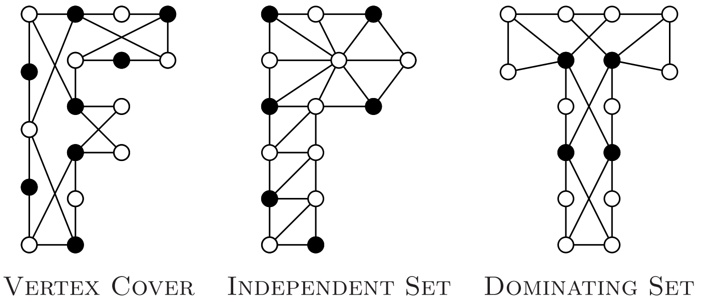

Figure 1.1: Examples for VERTEX COVER (left-hand graph), INDEPENDENT SET (middle graph), and DOMINATING SET (right-hand graph). In each graph, the black vertices form an optimal solution for the corresponding optimization problem. Note that all graphs are planar.

- The problems are central in the study of parameterized complexity since they are (the most intensively studied) representatives for the three classes

$$\mathrm{FPT},\ \mathrm{W}[1],\ \text{and}\ \mathrm{W}[2].$$

  More precisely, par-VERTEX COVER is in FPT; par-INDEPENDENT SET is complete for W[1]; and par-DOMINATING SET is complete for W[2].

1.3.1 Vertex Cover

The VERTEX COVER problem is probably one of the most prominent graph problems in both classical and parameterized complexity study. On the one hand, it was among the first graph problems for which NP-completeness was proved. On the other hand, reviewing recent papers on parameterized complexity issues, there is no denying the fact that par-VERTEX COVER is the standard problem for an introduction to the theory of fixed-parameter algorithms.

Definition 1.3.3
VERTEX COVER *is the optimization problem* $(\mathsf{I}, \mathsf{F}, \mathsf{c}, \min)$ *which is defined as follows:*

(i) *The set of instances* I *consists of all (undirected) graphs* $\mathsf{G} = (\mathsf{V}, \mathsf{E})$.

(ii) *A feasible solution is a subset* $\mathsf{V}' \subseteq \mathsf{V}$ *which* covers *all edges, i.e.,* $\forall \mathsf{e} \in \mathsf{E} : \mathsf{V}' \cap \mathsf{e} \neq \emptyset$. *We call such a set a* vertex cover.

(iii) *The cost or* size *for a feasible solution* V' *is given by* $\mathsf{c}(\mathsf{V}') := |\mathsf{V}'|$.

We denote by $\mathrm{vc}(\mathsf{G})$ *the size of the smallest vertex cover of a graph* G*, and call this number the* vertex cover number.

An example for VERTEX COVER is shown in Fig. 1.1 (left-hand graph).

The core application of the VERTEX COVER problem arises in the field of resolving so-called conflict graphs. In a conflict graph, vertices represent parties and edges model conflicts between these parties. The aim in such conflict-graphs is to remove as few parties as necessary to resolve all conflicts. As reported in [156], applications for this problem appear, e.g., in computational biology. Among other applications we want to highlight the spare allocation problem in the area of reconfigurable VLSI design [95, 133].

VERTEX COVER is known to be NP-complete by a reduction from 3 SAT [124]. The problem is approximable within $2 - \frac{\log\log|V|}{2\log|V|}$ [33, 151]. Observe that this is only a slight improvement on a trivial 2-approximation algorithm which greedily considers an arbitrary edge in the current graph, takes both endpoints of the edge in the vertex cover to be constructed, and removes these vertices from the graph. There is still a certain gap to the best known lower bound due to Håstad [108] who showed that an approximation-factor of 1.1666 cannot be achieved unless $P = NP$. For further results on the approximability of VERTEX COVER we refer to [68].

When restricted to planar graphs, the problem still remains NP-complete even if we require that no vertex degree exceeds four [100]. However, a polynomial time approximation scheme (PTAS) is given for VERTEX COVER on planar graphs in Baker's well-known work [31]. That is, there is a polynomial time approximation algorithm with approximation factor $1 + \varepsilon$, where ε is a constant arbitrarily close to 0. The degree of the polynomial grows with $1/\varepsilon$. We mention in passing, that this PTAS is extendable to all classes of graphs that exclude a minor [104].

Let us turn our attention to the parameterized complexity of the problem. The parameterized problem par-VERTEX COVER (see Definition 1.3.2) is in FPT and probably the most often cited prototype example for this class with a long history on steadily improved fixed-parameter algorithms [82, p.5]. The first fixed-parameter algorithm was non-constructive and derived from the heavy machinery of graph minor theory [93]. Later, it was observed that based on a similar idea as the aforementioned 2-approximation algorithm an easy $O(2^k n)$ search tree can be constructed. Since then, in the literature, we find a whole sequence of contributions [32, 180, 85, 156, 56] employing more and more fine-grained search tree algorithms (see Section 3.1.2.1). The currently best known algorithm has running time $O(1.2852^k + kn)$ [56] using a combination of bounded search tree and problem kernel reduction. It was unclear whether, for planar graphs, a better algorithm is possible. We provide new results in this direction in Chapters 4 and 5.

1.3.2 Independent Set

Next, we introduce a closely related maximization problem.

Definition 1.3.4
INDEPENDENT SET *is the optimization problem* $(I, F, c, \max)$ *which is defined as follows:*

(i) *The set of instances* I *consists of all (undirected) graphs* $G = (V, E)$.

(ii) *A feasible solution is a subset* $V' \subseteq V$ *which is* independent*, i.e.,* $\forall v, w \in V' : \{v, w\} \notin E$*. We call such a set an* independent set*.*

(iii) *The cost or* size *for a feasible solution* V' *is given by* $c(V') := |V'|$.

We denote by is(G) *the size of the largest independent set of a graph* G*, and call this number the* stability number*.*

An example for INDEPENDENT SET is shown in Fig. 1.1 (middle graph).

As in the case for VERTEX COVER this problem has its applications mainly in resolving conflict graphs. However, since this is a maximization problem, the goal is to keep as many items as possible that are mutually non-conflicting. Concrete applications arise, e.g., in the field of frequency assignment problems [142] (see Section 4.3), or in the area of map labeling problems [2]. Here, the goal is to maximize the number of labels (e.g., names of cities, streets,...) on a map—each label is given together with suitable positions—such that no two labels overlap.

INDEPENDENT SET is closely related to VERTEX COVER: Observe that for each graph it holds that the complement vertex set of a (minimum) vertex cover is a (maximum) independent set. This is the key argument behind the NP-completeness proof for INDEPENDENT SET [100]. Moreover, there is a close relation to the CLIQUE problem[7]: The vertices of a (maximum) independent set form a (maximum) clique in the complementary graph and vice versa. Hence, concerning approximation results, INDEPENDENT SET enjoys the same properties as CLIQUE, e.g., there is an $O(\frac{|V|}{(\log|V|)^2})$-approximation algorithm due to Boppana and Halldórsson [48] and the problem is not approximable within $|V|^{1/4-\varepsilon}$ for any $\varepsilon > 0$ unless P = NP [34].

It is known that the parameterized version par-INDEPENDENT SET (see Definition 1.3.2) is complete for the class W[1] [81, 82]. We want to emphasize that, from a parameterized point of view, there is a subtle but crucial difference between the aforementioned relations of par-INDEPENDENT SET to par-VERTEX COVER and to par-CLIQUE, respectively. On the one hand, for a graph $G = (V, E)$, we obtain $\mathrm{vc}(G) = |V| - \mathrm{is}(G)$ which means that the above given reduction from par-INDEPENDENT SET to par-VERTEX COVER is not "parameter-preserving"—the new problem parameter is(G) depends on the *size* of the graph and not only the old parameter vc(G). In fact, par-VERTEX COVER and par-INDEPENDENT SET are so-called *dual problems*. If a *parameterized reduction*[8] from par-INDEPENDENT SET to par-VERTEX COVER existed then the W-hierarchy would collapse at the lowest level, i.e., then FPT = W[1]. On the other hand, transforming a graph G to its complementary graph G^c *is* a parameterized reduction from par-INDEPENDENT SET to par-CLIQUE and vice versa: letting cl(G) denote the size of a maximal clique in G, we have $\mathrm{is}(G^c) = \mathrm{cl}(G)$. Hence, par-INDEPENDENT SET and par-CLIQUE both are equally hard from a parameterized point of view.[9]

Restricting to planar graphs INDEPENDENT SET is still NP-complete even if the graph is cubic [100]. Baker [31] gave a PTAS for INDEPENDENT SET on planar graphs, which again is extendable to all classes of graphs that exclude a minor [104]. The parameterized problem par-INDEPENDENT SET on planar graphs is no longer W[1]-hard but admits a fixed-parameter algorithm. For a simple search tree algorithm running in time $O(6^k n)$, we refer to Section 3.1.2.2.

[7]In the CLIQUE problem one is given an undirected graph and the task is to find a maximum set of vertices that form a clique, i.e., that are pairwisely joined by edges.

[8]For a precise definition on parameterized reductions, we refer to Definition 4.2.8.

[9]The reason why, in this work, we focus on par-INDEPENDENT SET instead of par-CLIQUE is due to the fact that we focus our attention on planar graph problems. It is clear, that par-CLIQUE does not make sense in this setting, since a planar graph has no clique larger than four.

Further algorithms for par-INDEPENDENT SET will be given in Chapters 4 and 5.

1.3.3 Dominating Set

The DOMINATING SET problem is among the core problems in algorithms, combinatorial optimization, and computational complexity [30, 68, 100, 116, 161]. Solving domination-like problems developed into a research area on its own (see [109, 110, 111, 112] for comprehensive overviews). According to a 1998 survey [110, Preface], more than 1200 research papers were published on domination in graphs, among which more than 200 publications and more than 30 PhD theses investigate the algorithmic complexity of domination and related problems [110, Chapter 12].

Definition 1.3.5
DOMINATING SET *is the optimization problem* $(I, F, c, \min)$ *which is defined as follows:*

(i) *The set of instances* I *consists of all (undirected) graphs* $G = (V, E)$.

(ii) *A feasible solution is a subset* $V' \subseteq V$ *which* dominates *all vertices in* $V \setminus V'$*, i.e.,* $V \subseteq N[V']$*. We call such a set a* dominating set.

(iii) *The cost or* size *for a feasible solution* V' *is given by* $c(V') := |V'|$.

We denote by $\mathrm{ds}(G)$ *the size of the smallest dominating set of a graph* G*, and call this number the* domination number.

An example for DOMINATING SET is shown in Fig. 1.1 (right-hand graph).

Domination problems occur in numerous practical settings. Typical applications are facility location problems as presented at the beginning of this chapter. These imply strategic decision problems, such as, e.g., locating base stations in a mobile network such that each customer is served [99, 102], placing servers in a network [160], positioning security system cameras to observe an environment [35], or placing emergency service stations. Further applications comprise models in social network theory [125], routing in wireless networks [190], game theory [154], computational biology, or voting systems [106] (see [110, 111, 169] for a survey).

DOMINATING SET is known to be NP-complete [100]. The approximability of the DOMINATING SET problem has received considerable attention [30, 68, 116]. It is known that DOMINATING SET is polynomial time approximable with factor $1 + \log |V|$ since the problem is a special case of the minimum SET COVER problem. On the negative side, however, it is not known and it is not believed that DOMINATING SET for general graphs has a constant factor approximation algorithm (see, e.g., [30, 68] for details). More precisely, if $P \neq NP$, then DOMINATING SET has no polynomial time approximation scheme, see [27, p. 503], since it is hard for the approximation class MAXSNP defined in [162]. Feige [91, p. 637] has even shown that the problem is not approximable within $(1 - \varepsilon) \ln |V|$ for any $\varepsilon > 0$ unless $NP \subseteq DTIME(n^{\log \log n})$ [91].

In the case of planar graphs, the problem remains NP-complete, even when restricted to planar graphs with maximum vertex degree 3 and to planar graphs that are regular of degree 4 [100].

Baker [31] states that the shifting techniques used to derive the PTAS for INDEPENDENT SET on planar graphs can be carried over to DOMINATING SET on planar graphs.[10] Again, Grohe extended this result to graphs that exclude a minor [104]. For further complexity results for DOMINATING SET, we refer to [110, 140, 164]. We mention one interesting fact concerning complete grid graphs. Unlike VERTEX COVER and INDEPENDENT SET which are solvable in linear time on a complete $n \times m$-grid $P_n \times P_m$, DOMINATING SET happens to be NP-complete [62]. There is a series of papers that investigate and compute the domination number $\mathrm{ds}(P_n \times P_m)$ for a small fixed n in a certain range of m [55, 107, 178] and try to detect certain periodicities in these numbers [54, 63, 120, 139].

What is known about the parameterized complexity of the par-DOMINATING SET problem (see Definition 1.3.2)? On general graphs par-DOMINATING SET is W[2]-complete [79, 82], i.e., we believe that the complexity of handling the parameter is even more sophisticated than in the case of the W[1]-complete par-INDEPENDENT SET problem. On planar graphs, Downey and Fellows [80, 82] claimed a search tree algorithm for par-DOMINATING SET running in time $O(11^k n)$. The analysis of their algorithm, however, turned out to be flawed. We give a corrected *and* improved search tree algorithm in Chapter 3. Besides, further improved fixed-parameter algorithms will be presented in Chapters 4 and 5. We mention in passing that, from the general context of model checking on structures that are locally tree-decomposable, it can be derived that par-DOMINATING SET on planar graphs is fixed-parameter tractable [98, Theorem 1.1]. However, it seems that this general context only serves as classification tool not yielding efficient algorithms.

1.3.4 Further Problems

In the following, we summarize some important modifications and variants of the DOMINATING SET problem. Haynes *et al.* [110, p. 327 ff] name more than 75 related problems among which we concentrate on a small selection. A *property* P of a vertex set $V' \subseteq V$ of an undirected graph $G = (V, E)$ will be a Boolean predicate which yields true or false values when given as input V, E, and V'. Since V and E will always be clear from the context, we will simply write $P(V')$ instead of $P(V, E, V')$. The DOMINATING SET WITH PROPERTY P problem is defined analogously to DOMINATING SET (see Definition 1.3.5) with the exception that a feasible solution for a graph $G = (V, E)$ now is a dominating set D with property P, i.e., such that $P(D)$ is true.

Examples for such problems (all also appearing in [110, 132, 183, 184]) are:

- the INDEPENDENT DOMINATING SET problem, where the property $P(D)$ of the dominating set D is that D is independent,
- the TOTAL DOMINATING SET problem, where the property $P(D)$ of the dominating set D is that each vertex of D has a neighbor in D,
- the PERFECT DOMINATING SET problem, where the property $P(D)$ is that each vertex which is not in D has *exactly* one neighbor in D,
- the PERFECT CODE problem, where the dominating set has to be perfect and independent, and

[10] A small flaw in the running time stated in [31] was detected and corrected in [123].

- the TOTAL PERFECT DOMINATING SET problem, where the dominating set has to be total and perfect.

All mentioned problems are NP-complete which follows from a general framework on domination-like problems due to Telle [181]. For an overview on the complexity of some of these problems restricted to special graph classes we refer to [132, Fig. 8.3]. Finally, we will deal with the NP-complete FACE COVER problem which is defined as follows (see [39, 82, 166]).

Definition 1.3.6
FACE COVER *is the optimization problem* $(\mathsf{I}, \mathsf{F}, \mathsf{c}, \min)$ *which is defined as follows:*

(i) *The set of instances* I *consists of all plane graphs* $(\mathsf{G} = (\mathsf{V}, \mathsf{E}), \phi)$.

(ii) *A feasible solution is a subset* F' *of the set of faces* F *which covers all vertices, i.e., each vertex in* V *is on the boundary of at least one face in* F'. *We call such a set a* face cover.

(iii) *The cost or* size *for a feasible solution* F' *is given by* $\mathsf{c}(\mathsf{F}') := |\mathsf{F}'|$.

1.4 Overview of New Results

In the following we give a short chapterwise summary of the new results covered by this thesis. For details we refer to the corresponding chapters.

Data reduction (Chapter 2)

Description. Data reduction is important in theory as well as in practice. In parameterized complexity data reduction mainly is achieved by so-called problem kernelization. From a theoretical point of view, the existence of a problem kernel for a parameterized problem is equivalent to the fixed-parameter tractability of the problem, thus, yielding a different characterization of the class FPT. From a practical point of view, data reduction can be an extremely powerful tool: Enriching a known algorithm with preprocessing by problem kernelization may lead to enormous speed-ups. We underpinned this by various experimental studies demonstrating the practical strength of data reduction.

Results. We revisit the linear problem kernel due to Nemhauser/Trotter for par-VERTEX COVER and then prove linear problem kernels for two further parameterized problems. Firstly, we make an easy observation based on the four color theorem that establishes a linear problem kernel for par-INDEPENDENT SET on planar graphs. Secondly, as our main result, we prove a linear problem kernel for par-DOMINATING SET on planar graphs, thus affirmly answering an open question from previous work. The result is based on two simple reduction rules, yet, the proof involves complicated combinatorial arguments. This problem kernelization proved to be very efficient in practice: a set of combinatorially generated random planar graphs was reduced by more than 99%. Thus, we come up with a practically promising as well as theoretically appealing result for computing the domination number of a graph, one of the so far few positive news for this important problem.

Search tree algorithms (Chapter 3)

Description. Constructing a bounded search tree is probably the most wide-spread technique in the design of fixed-parameter algorithms. Here, we consider a special form of search tree algorithms for combinatorial graph problems called the degree-branching method.

Results. We revisit the degree-branching method for par-VERTEX COVER and for par-INDEPENDENT SET on planar graphs. The key result in this chapter is to provide and, in particular to analyze, a time $O(8^k n)$ degree-branching algorithm for the much more involved par-DOMINATING SET problem on planar graphs. Here k is the size of the dominating set we seek for. To establish this result we prove an intricate branching theorem based on the Euler formula. Our theorem states that every planar black and white graph which is reduced with respect to certain simple reduction rules admits a black vertex of degree at most seven (which then is used as a branching vertex). Since a previous result claiming a time $O(11^k n)$ algorithm [82, Theorem 3.4] happens to contain an obvious flaw, our result seems to give the first completely correct analysis for an par-DOMINATING SET algorithm on planar graphs with running time $O(c^k n)$ for small constant c that even improves on the previously claimed constant considerably.

Graph separation (Chapter 4)

Description. Many hard combinatorial graph problems can be attacked by a divide-and-conquer strategy if the input instances are taken from a graph class that admits a (vertex) separator theorem. We coin the notion of "glueability" to characterize such graph problems. Throughout this chapter, we design fixed-parameter algorithms by combining (linear) problem kernelization with the aforementioned divide-and-conquer approach based on graph separation.

Results. This chapter has two main objectives. Firstly, we focus on graph classes $\mathbb{G}$ for which a so-called $\sqrt{\cdot}$-separator theorem is known, as, e.g., in the case of planar graphs or, more generally, on the class of graphs of bounded genus. We give a general methodology how to derive time $2^{O(\sqrt{k})} + n^{O(1)}$ algorithms for glueable parameterized graph problems which admit a linear problem kernel on $\mathbb{G}$. Here n is the size of the input graph, and k is the corresponding problem parameter. These include par-VERTEX COVER, par-INDEPENDENT SET, and par-DOMINATING SET on planar graphs. In this way we gain an exponential improvement over previous exact solutions for the problems. The claimed algorithms seem to be the first fixed-parameter algorithms at all with a *sublinear* exponent in the exponential running time component. Moreover, we show that their running time is asymptotically optimal by providing a lower bound of the following form: If any of the three problems par-VERTEX COVER, par-INDEPENDENT SET, or par-DOMINATING SET can be solved in time $2^{o(\sqrt{k})} n^{O(1)}$ on planar graphs, then $3\,\text{SAT} \in \text{DTIME}(2^{o(n)})$, where (in the latter case) n is the number of variables of a 3 SAT formula. This fact, i.e., that 3 SAT can be solved in exponential time with a sublinear exponent, is generally considered to be unlikely in classical complexity theory.

Secondly, we extend the aforementioned methods to the class of disk intersection graphs for which a (classical) $\sqrt{\cdot}$-separator theorem does not exist. The key contribution here is to prove a new geometric version of a $\sqrt{\cdot}$-separator theorem for disk graphs of bounded radius ratio. As an application we consider the par-INDEPENDENT SET problem on this graph class which is motivated by modeling conflict graphs that appear, e.g., when studying interference problems in frequency assignment. We obtain a time $2^{O(\sqrt{k}\log(n))}$ algorithm for this problem. Such a sublinear exponent cannot be obtained for par-INDEPENDENT SET on general graphs unless $3\,\text{SAT} \in \text{DTIME}(2^{o(n)})$, where n is the number of variables in a 3 SAT formula. In the case of so-called ϑ-precision disk graphs we expose algorithms for par-INDEPENDENT SET and par-DOMINATING SET with running time $O(2^{O(\sqrt{k})} + n)$, which establishes fixed-parameter tractability.

Tree decompositions (Chapter 5)

Description. Tree decomposition based algorithms for (hard) combinatorial graph problems generally consist of two phases: constructing a tree decomposition of the given input graph, and then solving the underlying graph problem using dynamic programming on the tree decomposition. We exhibit the generality and the usefulness of a tree decomposition based approach in the case of parameterized planar graph problems. For that purpose, we introduce the novel so-called "Layerwise Separation Property" (LSP) as a key unifying tool to generically obtain time $O(2^{O(\sqrt{k})}n)$ algorithms.

Results. Firstly, we focus on the construction of tree decompositions. We prove a structural relationship for the treewidth $\text{tw}(G)$ of a planar graph G stating that $\text{tw}(G) \leq c \cdot \sqrt{k}$, for every "yes"-instance (G, k) of an LSP-problem.[11] Moreover, we demonstrate how to construct the corresponding tree decompositions in time $O(\sqrt{k} \cdot n)$. Specializing to VERTEX COVER or DOMINATING SET, we obtain new upper bounds relating the treewidth of a planar graph G with its vertex cover number $\text{vc}(G)$ or the domination number $\text{ds}(G)$, namely: $\text{tw}(G) \leq 4\sqrt{3\,\text{vc}(G)} + 5$ and $\text{tw}(G) \leq 6\sqrt{34\,\text{ds}(G)} + 8$. Both upper bounds are asymptotically optimal.

Secondly, we considerably improve upon the previously best known dynamic programming algorithms for various domination-like problems on graphs of bounded treewidth. E.g., we demonstrate how DOMINATING SET can be solved in time $O(4^{\ell}N)$ (here, ℓ being the upper bound on the treewidth and N being the number of tree nodes in the underlying tree decomposition) whereas the state of the art had been $O(9^{\ell}N)$ [183, 184]. Similar improvements are given for a series of problems, among others for VERTEX COVER, INDEPENDENT DOMINATING SET, TOTAL DOMINATING SET, PERFECT DOMINATING SET, or PERFECT CODE.

Putting the two phases together we, thus, establish a universal scheme to obtain time $O(2^{O(\sqrt{k})}n)$ algorithms for all LSP-problems that can be solved by dynamic programming on a given tree decomposition.[12] We provide formulas on how to compute the constants

[11] The constant c can be computed from the so-called width and size-factor of the LSP-problem.

[12] Most of the algorithms can be sped-up to $2^{O(\sqrt{k})} + n^{O(1)}$ using problem kernelization.

hidden in the exponent from various problem-specific parameters. In particular, we come up with a time $O(2^{4\sqrt{3k}}k + kn)$ algorithm for par-VERTEX COVER, an $O(2^{4\sqrt{6k}}k + n^2)$ algorithm for par-INDEPENDENT SET, and an $O(2^{12\sqrt{17\log(3)k}}k + n^3)$ algorithm for par-DOMINATING SET on planar graphs. Here n is the size of the input graph and k is the corresponding parameter (i.e., the size of the vertex cover, the independent set, or the dominating set we seek for). Similar results hold true for various variations of these problems such as several par-DOMINATING SET WITH PROPERTY P problems or par-FACE COVER. These algorithms outperform the ones obtained by graph separation and linear problem kernelization. In addition, the Layerwise Separation Property is seemingly less restrictive than requiring a linear problem kernel (as needed for the graph separation approach).[13] In this sense, the tree decomposition based approach applies to a broader class of parameterized planar graph problems than the graph separation approach.

Experimental Studies (Chapter 6)

Description. Most of the given algorithms were implemented and included in our "FPT-Toolbox" an easy to use software package designed to compute exact solutions for various NP-hard planar graph problems. Based on these implementations, we report on a first serious round of experiments using combinatorially generated random graph samples. The results are very encouraging.[14]

Results. Our empirical studies reveal that, in all examined cases, the algorithms were competitive and the actual running time behavior is far from the corresponding worst-case performance bounds that were derived from the theoretical analysis. We highlight some of the findings: For the par-DOMINATING SET search tree algorithm, we detect that the actual average branching degree is around 1.6 with a measured worst case branching degree of 4, whereas the (tight) worst case analysis stated a branching degree of 8. For the tree decomposition based algorithms, we measure, among others, that the width obtained for the tree decompositions is considerably smaller than the one expected by our worst-case analysis which results in much smaller overall running times than predicted. The most significant finding concerns the power of data reduction through problem kernelization: e.g., the data reduction for VERTEX COVER due to Nemhauser and Trotter shrinks, on our set of test instances, the input graphs by around 65%; our new kernelization rules for DOMINATING SET succeed in a reduction of the input graphs by more than 99%. Incorporating data reduction strategies results in enormous speed-ups of all our algorithms.

[13] We provide evidence for this by some examples for LSP-problems for which a linear problem kernel so far is unknown.

[14] We remark that our investigations are based on randomly generated graphs only, and that our findings might, therefore, not straight-forwardly be carried over to problem instances arising from specific applications. Still, our tests indicate the practical significance and competitive capacity of our algorithms.

Chapter 2

Data Reduction

Data reduction by preprocessing is a widely used core technique in algorithm engineering [52]; it is perhaps one of the most beneficial strategies to solve hard problems in practice. Weihe [188, 189] gave a striking example when dealing with the NP-complete RED-BLUE DOMINATING SET problem in context of the European railroad network. In a preprocessing phase, he applied two simple data reduction rules again and again until no further application was possible. The impressive result of his empirical study was that each of his real-world instances was broken into very small pieces such that for each of these a simple brute-force approach was sufficient to solve the hard problems efficiently and optimally.

In the development of fixed-parameter algorithms data reduction is achieved by so-called problem kernel reduction. In a sense, the idea behind is to cut off the "easy parts" of a given problem instance such that only the "hard kernel" of the problem remains, where then, e.g., exhaustive search can be applied (with reduced costs). To get a problem kernel as small as possible is, therefore, a central goal from a practical as well as from a theoretical point of view.

In Section 2.1, we give a short introduction to the concept of "reduction to problem kernel." Besides we will enrich the theory with two examples for linear problem kernels, namely par-INDEPENDENT SET on planar graphs and par-VERTEX COVER. A linear problem kernel for the par-DOMINATING SET problem on planar graphs is shown in Section 2.2, thus, answering a question that was open for the last few years.

2.1 Background

This section provides a formal definition of reduction to problem kernel, demonstrates the relevance of this technique to the design of fixed-parameter algorithms and gives first examples for so-called linear problem kernels.

2.1.1 The Concept of Data Reduction by (Linear) Problem Kernels

2.1.1.1 Definition and another characterization of FPT

Definition 2.1.1 *Let $\mathcal{L} \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem. We say that $\mathcal{L}$ is* kernelizable *(or that $\mathcal{L}$* admits a problem kernel*) if there exists a mapping*

$$\Phi : \Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N}, \quad (x, k) \mapsto (x', k'),$$

and an arbitrary function $g : \mathbb{N} \to \mathbb{N}$ such that

(i) *Φ is computable in time p_{kernel} polynomial in $|x|$ and k.*[1]

(ii) *$(x, k) \in \mathcal{L}$ if and only if $(x', k') \in \mathcal{L}$.*

(iii) *$|x'| \leq g(k')$ and $k' \leq k$.*

The instance x' is called the reduced instance *or the* problem kernel *and $g(k')$ is called the* size *of the problem kernel. If g is a linear function then we simply say that $\mathcal{L}$* admits a linear problem kernel. Reduction to problem kernel *means replacing (x, k) by (x', k').*

It is clear that a kernelizable parameterized problem $\mathcal{L}$ which can be solved in time $h(|x|, k)$ for an input instance (x, k) admits a solving algorithm running in time $O(h(g(k), k) + p_{\text{kernel}}(|x|, k))$, and hence is in FPT. The following theorem shows that the converse is also true, i.e., that kernelizability exactly characterizes the class FPT.

Theorem 2.1.2 *Let $\mathcal{L}$ be a parameterized problem which is decidable. Then $\mathcal{L} \in$ FPT if and only if $\mathcal{L}$ is kernelizable.*

Proof: In the discussion before this theorem we have already shown one direction. It remains to prove kernelizability of $\mathcal{L}$ assuming $\mathcal{L} \in$ FPT Suppose we have a time $f(k)n^\alpha$ algorithm A to solve the word problem for $\mathcal{L}$. For an instance (x, k), we do the following: Run A on (x, k) for $|x|^{\alpha+1}$ steps.

Case 1: A stops (i.e., $|x|^{\alpha+1} \geq f(k)|x|^\alpha$) and returns the answer whether $(x, k) \in \mathcal{L}$ or not. In this case do a constant reduction, i.e.,

$$\Phi((x, k)) := \begin{cases} (x_0, k_0) & \text{if } (x, k) \in \mathcal{L}, \\ (x_1, k_1) & \text{if } (x, k) \notin \mathcal{L}, \end{cases}$$

for some $(x_0, k_0) \in \mathcal{L}$, and $(x_1, k_1) \notin \mathcal{L}$ (with $|x_0| \leq f(k_0)$ and $|x_1| \leq f(k_1)$).

Case 2: A does not return a result, but this means, that

$$|x|^{\alpha+1} < f(k)|x|^\alpha,$$

which is equivalent to $|x| < f(k)$. In this case let $\Phi((x, k)) := (x, k)$.

Then Φ yields the desired reduction to problem kernel. □

It must be emphasized that Theorem 2.1.2 is interesting only from a theoretical point of view, whereas all other results in this chapter are also significant from a practical point of view.

[1] In principal, we could also allow a running time of the form $f(k) \cdot |x|^{O(1)}$, where f is an arbitrary function.

2.1.1.2 The importance of (small) problem kernels.

We have demonstrated the importance of reduction to problem kernel in theory. There are various other aspects which underpin the central role of this technique. Throughout this work, we will see close relations of reduction to problem kernel to basically all chapters.

Reduction to problem kernel as preprocessing (see Chapter 6). Reduction to problem kernel itself can be seen as a preprocessing step for hard problems: From a given instance (x, k), in a preprocessing phase, one computes the problem kernel (x', k') in polynomial time and then solves the problem for the kernel. It is clear that the size of the problem kernel has a significant impact on the overall running time. Hence, small problem kernels are desirable. In this sense, Theorem 2.1.2 is of theoretical importance only, since the guaranteed size of a problem kernel for a parameterized problem that can be solved in time $f(k) \cdot n^{O(1)}$ is $f(k)$, which is exponential in general. It is a challenging and algorithmically important task to find problem kernels which are small, where by "small" we mean of polynomial or even linear size in k.

Reduction to problem kernel and bounded search trees (see Chapter 3). Niedermeier and Rossmanith [157] showed an interesting technique to interleave reduction to problem kernel with bounded search trees (see Chapter 3 for details on bounded search trees). More specifically, they proposed a method to speed up bounded search tree algorithms for parameterized problems that admit a problem kernel of polynomial size.

Theorem 2.1.3 *Let $\mathcal{L}$ be a parameterized problem which can be solved in time $O(c^k n)$ by a search tree algorithm. If $\mathcal{L}$ admits a problem kernel of polynomial size $g(k)$ that can be computed in time $p_{kernel}(n, k)$ then we can solve $\mathcal{L}$ in time $O(c^k + p_{kernel}(n, k))$.* □

Note that by the observations before Theorem 2.1.2 a simple combination of reduction to problem kernel and bounded search tree algorithm would yield time complexity $O(c^k g(k) + p_{\mathrm{kernel}}(n, k))$. The key contribution of [157] is to get rid of the polynomial factor $g(k)$. This can be achieved by a thorough analysis of the modified search tree algorithm in which problem kernel reduction is performed to the current instance in *sufficiently many* search tree nodes.

Reduction to problem kernel and fixed-parameter algorithms with sublinear exponents (see Chapters 4 and 5). A recent achievement in the theory of fixed-parameter tractability is the design of time $f(k) \cdot n^{O(1)}$ algorithms where the running time component f grows sublinearly in the exponent, i.e., is of the form $f(k) = 2^{O(\sqrt{k})}$ (see Chapters 4 and 5). There are basically two distinct strategies—both being in a sense generic—to obtain algorithms for various planar graph problems with this running time behavior. Reduction to problem kernel plays a central role for both strategies. One method is to combine reduction to problem kernel with divide-and-conquer algorithms based on graph separation (see Section 4.2) and the other one is a tree decomposition based approach. The key to the latter method is the so-called "Layerwise Separation Property" for which kernelizability (with a linear problem kernel) is a sufficient condition (see Section 5.4 for details).

In the remainder of this section we will revisit the threesome par-VERTEX COVER, par-INDEPENDENT SET and par-DOMINATING SET (on planar graphs) with respect to kernelizability and possible "small" problem kernels.

2.1.2 Nemhauser and Trotter's Theorem and other Examples

Often the best one can hope for is that the problem kernel has size linear in k, a so-called *linear problem kernel*. In the following, we, firstly, present some easy observations which establish a linear problem kernel for the par-INDEPENDENT SET problem on planar graphs. Secondly, we cite a theorem of Nemhauser and Trotter [153], (also refer to [33, 163]), from which Chen *et al.* [56] recently derived a linear problem kernel for par-VERTEX COVER on general (not necessarily planar) graphs.

By making use of the four color theorem for planar graphs and the corresponding algorithm generating a four coloring [170], the following result can be obtained easily.

Proposition 2.1.4 par-INDEPENDENT SET *on planar graphs admits a linear problem kernel of size* $4k$. *The reduction to problem kernel can be carried out in time* $O(n^2)$.

Proof: Due to the four-color theorem (see [21, 22] or [170]), every maximum independent set of a planar graph $G = (V, E)$ with n vertices has size at least $\lceil n/4 \rceil$: Let $V = V_1 \dot{\cup} \cdots \dot{\cup} V_4$ be a four coloring, then every "color set" V_i is an independent set and there exists at least one such set V_{i_0} with $|V_{i_0}| \geq \lceil n/4 \rceil$. Hence,

$$\Phi((G,k)) := (G', k') := \begin{cases} (G,k) & \text{if } k > \lceil n/4 \rceil, \\ (G_0, k_0) & \text{otherwise,} \end{cases}$$

where (G_0, k_0) is some trivial "yes"-instance of par-INDEPENDENT SET (e.g., $(G[V_{i_0}], |V_{i_0}|)$), yields the desired problem kernel reduction. Note that, by construction, for the reduced graph $G' = (V', E')$, it holds $|V'| \leq 4k'$. Using the four-coloring algorithm by Robertson *et al.* [170] this reduction can be carried out in quadratic time. □

Remark 2.1.5 Using a linear time five-color algorithm, e.g., the one by Chiba *et al.* [60] we can even get a *linear* time reduction to problem kernel with a problem kernel of size $5k$.

There is an easy generalization of the above principle.

Remark 2.1.6 We say that a maximization problem $\mathcal{Q} = (I_{\mathcal{Q}}, S_{\mathcal{Q}}, c_{\mathcal{Q}}, \max)$ *admits solutions of guaranteed fraction* $1/d$, where d is some constant, if for $x \in I_{\mathcal{Q}}$ a feasible solution $y \in S_{\mathcal{Q}}(x)$ with $c_{\mathcal{Q}}(y) \geq |x|/d$ exists. Then, a similar idea as the one in the above proof shows that if a maximization problem $\mathcal{Q}$ admits solutions of guaranteed fraction $1/d$, then par-$\mathcal{Q}$ admits a linear problem kernel of size dk.

We have seen that INDEPENDENT SET on a planar graphs admits solutions of guaranteed fraction $1/4$. We briefly mention in passing that other examples with a guaranteed fraction solution exist, for which a linear problem kernel can be derived in a similar way: e.g., the par-MAXSAT problem parameterized by the number m of satisfied clauses admits a solution of guaranteed fraction $1/2$ (see [141]), and the so-called par-BETWEENNESS problem admits a solution of guaranteed fraction $1/3$ (see [61]).

We now turn our attention to VERTEX COVER, for which Buss and Goldsmith [51] gave a very easy problem kernel reduction which transforms an instance (G, k) in time $O(kn)$ into a reduced instance (G', k'), where G' has no more than k^2 edges (and, hence, at most $2k^2$ vertices). The idea is that if we are looking for a vertex cover of size at most k, then a vertex with degree $k+1$ *has* to be part of such a vertex cover because, otherwise, k vertices would not suffice to cover all edges emanating from v. This can further be improved using the following theorem of Nemhauser and Trotter [153] (also see [127] for a very recent account on this result).

Theorem 2.1.7 *Let* $G = (V, E)$ *be a graph. There is an algorithm that finds in time* $O(\sqrt{|V|} \cdot |E|)$ *disjoint sets* $C_0, V_0 \subseteq V$ *which have the following properties:*

(i) *If* $D \subseteq V_0$ *is a vertex cover of* $G[V_0]$ *then* $D \cup C_0$ *is a vertex cover of* G.

(ii) *There exists an optimal vertex cover* C^* *of* G *with* $C_0 \subseteq C^*$.

(iii) *It holds that* $\mathrm{vc}(G[V_0]) \geq |V_0|/2$.

The proof of the theorem relies on an elegant construction: For a given graph $G = (V, E)$, let $V' = \{v' \mid v \in V\}$ be a copy of the vertex set. Then, the bipartite graph

$$B = (V \cup V', E_B), \text{ where } E_B := \{\{v, w'\} \mid \{v, w\} \in E\}$$

is constructed. From an optimal vertex cover C_B (which can be computed easily in a bipartite graph using maximum matching), one derives the sets

$$\begin{aligned} C_0 &:= \{v \mid v \in C_B \text{ AND } v' \in C_B\}, \text{ and} \\ V_0 &:= \{v \mid v \in C_B \text{ XOR } v' \in C_B\}, \end{aligned}$$

for which Nemhauser and Trotter proved that these sets fulfill the claimed properties. For algorithmic purposes this means that, in order to compute an optimal vertex cover in G, we may start from the set C_0 and add an optimal vertex cover of the reduced graph $G[V_0]$.

As observed in [56], from the Theorem of Nemhauser and Trotter, we obtain the following.

Corollary 2.1.8 par-VERTEX COVER *admits a linear problem kernel of size* $2k$. *The reduction to problem kernel can be carried out in time* $O(k^3 + kn)$.

Proof: Using the notation from Theorem 2.1.7, we define

$$\Phi_{\mathrm{NT}}((G, k)) := \begin{cases} (G[V_0], k - |C_0|) & \text{if } |V_0| \leq 2k, \\ (\hat{G}, \hat{k}) & \text{otherwise,} \end{cases}$$

where $(\hat{G}, \hat{k}) \notin$ par-VERTEX COVER is some trivial "no"-instance with $|\hat{G}| \leq 2\hat{k}$. To proof that this indeed yields a reduction to problem kernel it remains to show that $(G, k) \in$ par-VERTEX COVER if and only if $\Phi_{\mathrm{NT}}((G, k)) \in$ par-VERTEX COVER. To see this consider two cases. Firstly, assume that $|V_0| > 2k$. But then $k < \mathrm{vc}(G[V_0]) \leq \mathrm{vc}(G)$, by property (iii) of Theorem 2.1.7, which means that $(G, k) \notin$ par-VERTEX COVER.

Secondly, assume that $|V_0| \leq 2k$. If $(G, k) \in$ par-VERTEX COVER, by property (ii) of Theorem 2.1.7, we can assume that there exists a vertex cover C^* of G, with $|C^*| \leq k$, such that $C_0 \subseteq C^*$. Since $C_0 \cap V_0 = \emptyset$, the set $D := C^* \setminus C_0$ is a vertex cover of $G[V_0]$ with $|D| \leq k - |C_0|$. Conversely, if $(G[V_0], k - |C_0|) \in$ par-VERTEX COVER and D is a vertex cover with $|D| \leq k - |C_0|$ then, by property (i) of Theorem 2.1.7, $D \cup C_0$ is a vertex cover of G with $|D \cup C_0| \leq k$.

Let Φ_{BG} be the aforementioned problem kernel reduction of Buss and Goldsmith [51]. Then, clearly, $\Phi := \Phi_{\mathrm{NT}} \circ \Phi_{\mathrm{BG}}$, also is a problem kernel reduction yielding a problem kernel of size $2k$. Letting $(G_1, k_1) := \Phi_{\mathrm{BG}}((G, k))$, the total running time for computing $\Phi((G, k))$ is $O(kn + \sqrt{|V(G_1)|}|E(G_1)|) = O(kn + k^3)$. □

According to the current state of knowledge, this is the best one could hope for par-VERTEX COVER on general graphs, because a problem kernel of size $(2 - \varepsilon)k$ with constant $\varepsilon > 0$ would imply a factor $2 - \varepsilon$ polynomial time approximation algorithm for VERTEX COVER, which would mean a major breakthrough in approximation algorithms for VERTEX COVER [116]. Remarkably, for VERTEX COVER on planar graphs, better approximation algorithms are known [33], as well as a polynomial time approximation scheme [31]. It remains an open question whether par-VERTEX COVER on planar graphs admits a problem kernel of size dk with some $d < 2$.

2.2 A Linear Problem Kernel for par-DOMINATING SET on Planar Graphs

We will now prove a linear problem kernel for the par-DOMINATING SET problem on planar graphs. This is the first result on problem kernel reduction for par-DOMINATING SET on planar graphs, answering an open question from previous work. Not even a polynomial problem kernel was known before. Unless FPT = W[2], a similar result is very unlikely to hold for general graphs. This is because par-DOMINATING SET is W[2]-complete on general graphs and the existence of a (linear) problem kernel implies membership of the class FPT (see Theorem 2.1.2).

More precisely, in this section, we will show the following result. We partly follow [7].

Theorem 2.2.1 par-DOMINATING SET *on planar graphs admits a linear problem kernel of size* $335k$. *The reduction to problem kernel can be carried out in time* $O(n^3)$.

The reduction to problem kernel is established by two "reduction rules" which are applied to the input graph. These reduction rules, which are designed for general and not necessarily planar graphs, are presented in Subsection 2.2.1. We then prove that a planar graphs that is reduced with respect to these reduction rules is small, i.e., has at most $O(k)$ vertices. The proof consists of two stages. In a first step, we try to find a so-called "maximal region decomposition" of the vertices of a reduced planar graph (see Subsection 2.2.2). In a second step, we show, on the one hand, that such a maximal region decomposition must contain all but $O(k)$ many vertices. On the other hand, we prove that such a region decomposition uses at most $O(k)$ regions, each of which having size $O(1)$ (see Subsection 2.2.3). Combining the results then yields an upper bound of the form $O(k)$ on the size of the problem kernel.

Remark 2.2.2 We remark that, at the current state of research, we do not know whether a linear problem kernel can be obtained for variants of the par-DOMINATING SET problem on planar graphs as well, e.g., for par-INDEPENDENT DOMINATING SET, par-TOTAL DOMINATING SET, par-PERFECT DOMINATING SET, PERFECT CODE, or TOTAL PERFECT DOMINATING SET. The reduction rules which will establish Theorem 2.2.1 are tailored toward par-DOMINATING SET only and a straightforward adaption to these variants seems difficult.

2.2.1 Reduction Rules

We present two reduction rules for DOMINATING SET which both are based on the same principle: The rules explore the local structures of the graph and try to replace them by simpler structures. For the first reduction rule, the local structure will be the neighborhood of a single vertex. For the second reduction rule, we will deal with the union of the neighborhoods of a pair of vertices.

2.2.1.1 The Neighborhood of a Single Vertex

Consider a vertex $v \in V$ of the given graph $G = (V, E)$. We partition the vertices of the open neighborhood $N(v)$ of v into three different sets:

- the "exit vertices" $N_{\text{exit}}(v)$, through which we can "leave" the closed neighborhood $N[v]$,
- the "guard vertices" $N_{\text{guard}}(v)$, which are neighbors of exit-vertices, and
- the "prisoner vertices" $N_{\text{prison}}(v)$, which are not direct neighbors of an exit vertex.

More formally, we define

$$\begin{aligned} N_{\text{exit}}(v) &:= \{u \in N(v) \mid N(u) \setminus N[v] \neq \emptyset\},^{2} \\ N_{\text{guard}}(v) &:= \{u \in N(v) \setminus N_{\text{exit}}(v) \mid N(u) \cap N_{\text{exit}}(v) \neq \emptyset\}, \\ N_{\text{prison}}(v) &:= N(v) \setminus (N_{\text{exit}}(v) \cup N_{\text{guard}}(v)). \end{aligned}$$

An example which illustrates the partitioning of $N(v)$ into the subsets $N_{\text{exit}}(v)$, $N_{\text{guard}}(v)$, and $N_{\text{prison}}(v)$ can be seen in the left-hand diagram of Fig. 2.1.

Our first reduction rule is based on the fact that we try to detect an optimal domination of the prisoner vertices $N_{\text{prison}}(v)$ within our local structure $N(v)$.

[2]For two sets X, Y, where Y is not necessarily a subset of X, we use the convention that $X \setminus Y := \{x \in X : x \notin Y\}$.

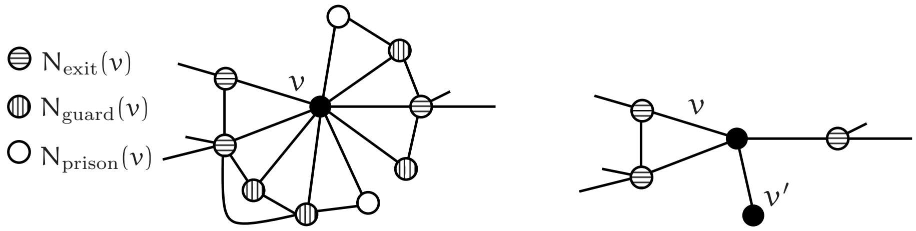

Figure 2.1: The left-hand side shows the partitioning of the neighborhood of a single vertex v. The vertices marked with horizontal lines are the exit vertices $N_{\text{exit}}(v)$. The guard vertices $N_{\text{guard}}(v)$ are marked with vertical lines. White vertices symbolize the prisoner vertices $N_{\text{prison}}(v)$. Since, for this example, $N_{\text{prison}}(v) \neq \emptyset$, reduction Rule 1 applies to v. The right-hand side shows the neighborhood after the application of Rule 1.

Rule 1 *If $N_{prison}(v) \neq \emptyset$ for some vertex v then*

- *remove $N_{guard}(v)$ and $N_{prison}(v)$ from G and*
- *add a new vertex v' with the edge $\{v, v'\}$.*

We use the vertex v' as a "gadget vertex" that enforces us to take v (or v') into an optimal dominating set in the reduced graph.

An example demonstrating the effect of this reduction rule is given in Fig. 2.1.

Lemma 2.2.3 *Let $G = (V, E)$ be a graph and let $G' = (V', E')$ be the resulting graph after having applied Rule 1 to G. Then $\text{ds}(G) = \text{ds}(G')$.*

Proof: Consider a vertex $v \in V$ such that $N_{\text{prison}}(v) \neq \emptyset$. The vertices in $N_{\text{prison}}(v)$ can only be dominated by either v or by vertices in $N_{\text{guard}}(v) \cup N_{\text{prison}}(v)$. But, clearly, $N(w) \subseteq N(v)$ for every $w \in N_{\text{guard}}(v) \cup N_{\text{prison}}(v)$. This shows that a best possible way to dominate $N_{\text{prison}}(v)$ is given by taking v into the dominating set. This is simulated by the "gadget" edge $\{v, v'\}$ in G'. It is safe to remove $N_{\text{guard}}(v) \cup N_{\text{prison}}(v)$, since $N(N_{\text{guard}}(v) \cup N_{\text{prison}}(v)) \subseteq N(v)$, i.e., since the vertices that could be dominated by vertices from $N_{\text{guard}}(v) \cup N_{\text{prison}}(v)$ are already dominated by v. Hence, $\text{ds}(G') = \text{ds}(G)$. □

Lemma 2.2.4 *Reduction Rule 1 can be carried out in time $O(n)$ for planar graphs and in time $O(n^3)$ for general graphs.*

Proof: The proof is deferred to the Appendix at the end of the chapter. □

2.2.1.2 The Neighborhood of a Pair of Vertices

Similar to Rule 1, we explore the union of the neighborhoods $N(v, w) := N(v) \cup N(w)$ of two vertices $v, w \in V$. Analogously, we now partition $N(v, w)$ into three disjoint subsets $N_{\text{exit}}(v, w)$ (the

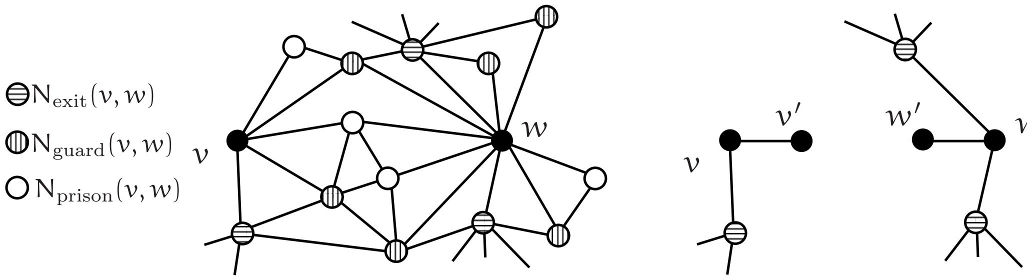

Figure 2.2: The left-hand side shows the partitioning of the neighborhood of a pair of vertices v and w. The vertices marked with horizontal lines are the exit vertices $N_{\text{exit}}(v, w)$. The guard vertices $N_{\text{guard}}(v, w)$ are marked with vertical lines. White vertices symbolize the prisoner vertices $N_{\text{prison}}(v, w)$. Since, for this example, $N_{\text{prison}}(v, w)$ cannot be dominated by a single vertex, Case 2 of Rule 2 applies. The right-hand side shows the neighborhood after the application of Rule 2.

"exit vertices" of $N(v, w)$), $N_{\text{guard}}(v, w)$ (the corresponding "guard vertices"), and $N_{\text{prison}}(v, w)$ (the "prisoner vertices" of $N(v, w)$). Setting $N[v, w] := N[v] \cup N[w]$, we define

$$\begin{aligned} N_{\text{exit}}(v, w) &:= \{u \in N(v, w) \mid N(u) \setminus N[v, w] \neq \emptyset\}, \\ N_{\text{guard}}(v, w) &:= \{u \in N(v, w) \setminus N_{\text{exit}}(v, w) \mid N(u) \cap N_{\text{exit}}(v, w) \neq \emptyset\}, \\ N_{\text{prison}}(v, w) &:= N(v, w) \setminus (N_{\text{exit}}(v, w) \cup N_{\text{guard}}(v, w)). \end{aligned}$$

The left-hand diagram of Fig. 2.2 shows an example which illustrates the partitioning of $N(v, w)$ into the subsets $N_{\text{exit}}(v, w)$, $N_{\text{guard}}(v, w)$, and $N_{\text{prison}}(v, w)$.

Our second reduction rule—compared to Rule 1—is slightly more complicated, but it is based on the same principle: We try to detect an optimal domination of the prisoner vertices $N_{\text{prison}}(v, w)$ in our local structure $N(v, w)$.

Rule 2 *Consider $v, w \in V$ ($v \neq w$) and suppose that $N_{prison}(v, w) \neq \emptyset$. Suppose that $N_{prison}(v, w)$ cannot be dominated by a single vertex from $N_{guard}(v, w) \cup N_{prison}(v, w)$.*

Case 1 *If $N_{prison}(v, w)$ can be dominated by a single vertex from $\{v, w\}$:*

(1.1) *If $N_{prison}(v, w) \subseteq N(v)$ as well as $N_{prison}(v, w) \subseteq N(w)$:*

- *remove $N_{prison}(v, w)$ and $N_{guard}(v, w) \cap N(v) \cap N(w)$ from G and*
- *add two new vertices z, z' and edges $\{v, z\}, \{w, z\}, \{v, z'\}, \{w, z'\}$ to G.*

(1.2) *If $N_{prison}(v, w) \subseteq N(v)$, but not $N_{prison}(v, w) \subseteq N(w)$:*

- *remove $N_{prison}(v, w)$ and $N_{guard}(v, w) \cap N(v)$ from G and*
- *add a new vertex v' and the edge $\{v, v'\}$ to G.*

(1.3) *If $N_{prison}(v, w) \subseteq N(w)$, but not $N_{prison}(v, w) \subseteq N(v)$:*

- *remove $N_{prison}(v, w)$ and $N_{guard}(v, w) \cap N(w)$ from G and*
- *add a new vertex w' and the edge $\{w, w'\}$ to G.*

Case 2 *If* $\mathsf{N}_{prison}(v, w)$ *cannot be dominated by a single vertex from* $\{v, w\}$*:*

- *remove* $\mathsf{N}_{prison}(v, w)$ *and* $\mathsf{N}_{guard}(v, w)$ *from* G *and*
- *add two new vertices* v', w' *and edges* $\{v, v'\}$, $\{w, w'\}$.

Again, the newly added vertices v' and w' of degree one act as gadgets that enforce us to take v or w into an optimal dominating set. A special situation is given in Case (1.1). Here, the gadget added to the graph G simulates that at least one of the vertices v or w has to be taken to an optimal dominating set.

The effect of this second reduction rule is demonstrated in Fig. 2.2 for the particular Case 2.

Lemma 2.2.5 *Let* $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ *be a graph and let* $\mathsf{G}' = (\mathsf{V}', \mathsf{E}')$ *be the resulting graph after having applied Rule 2 to* G*. Then* $\mathrm{ds}(\mathsf{G}) = \mathrm{ds}(\mathsf{G}')$.

Proof: Similar to the proof of Lemma 2.2.3, we observe that vertices from $\mathsf{N}_{\mathrm{prison}}(v, w)$ can only be dominated by vertices from $\mathsf{M} := \{v, w\} \cup \mathsf{N}_{\mathrm{guard}}(v, w) \cup \mathsf{N}_{\mathrm{prison}}(v, w)$. All cases in Rule 2 are based on the fact that $\mathsf{N}_{\mathrm{prison}}(v, w)$ needs to be dominated. Moreover, all cases only apply if there is not a *single* vertex in $\mathsf{N}_{\mathrm{guard}}(v, w) \cup \mathsf{N}_{\mathrm{prison}}(v, w)$ which dominates $\mathsf{N}_{\mathrm{prison}}(v, w)$.

We first of all discuss the correctness of Case (1.2) (and similarly obtain the correctness of the symmetric Case (1.3)): If v dominates $\mathsf{N}_{\mathrm{prison}}(v, w)$ (and w does not) then it is optimal to take v into the dominating set—and at the same time still leave the option of taking vertex w—instead of taking any combination of two vertices x and y from the set $\mathsf{M} \setminus \{v\}$. It may be that we still have to take w to a minimum dominating set, but in any case $\{v, w\}$ dominates at least as many vertices as x and y together. The "gadget edge" $\{v, v'\}$ simulates the effect of taking v. It is safe to remove $\mathsf{R} := (\mathsf{N}_{\mathrm{guard}}(v, w) \cap \mathsf{N}(v)) \cup \mathsf{N}_{\mathrm{prison}}(v, w)$ since, by taking v into the dominating set, all vertices in R are already dominated and since, as discussed above, it is always at least as good to take $\{v, w\}$ into a minimum dominating set than to take v and any other of the vertices from R.

In the situation of Case (1.1), we can dominate $\mathsf{N}_{\mathrm{prison}}(v, w)$ by either v or w. Since we cannot decide at this point which of these vertices should be chosen to be in the dominating set, we use the gadget with vertices z and z' which simulates a choice between v or w, as can be seen easily. In any case, however, it is at least as good to take one of the vertices v and w (maybe both) than to take any two vertices from $\mathsf{M} \setminus \{v, w\}$. The argument for this is similar to the one for Case (1.2). The removal of $\mathsf{N}_{\mathrm{prison}}(v, w) \cup (\mathsf{N}_{\mathrm{guard}}(v, w) \cap \mathsf{N}(v) \cap \mathsf{N}(w))$ is safe by a similar argument as the one that justified the removal of R in Case (1.2).

Finally, in Case 2, we clearly need at least two vertices to dominate $\mathsf{N}_{\mathrm{prison}}(v, w)$. Since $\mathsf{N}(v, w) \supseteq \mathsf{N}(x, y)$ for all pairs $x, y \in \mathsf{M}$ it is optimal to take v and w into the dominating set, simulated by the gadgets $\{v, v'\}$ and $\{w, w'\}$. As in the previous cases the removal of $\mathsf{N}_{\mathrm{prison}}(v, w) \cup \mathsf{N}_{\mathrm{guard}}(v, w)$ is safe since these vertices are already dominated and since these vertices need not be used for an optimal dominating set. □

Lemma 2.2.6 *Reduction Rule 2 can be carried out in time* $\mathsf{O}(n^2)$ *for planar graphs and in time* $\mathsf{O}(n^4)$ *for general graphs.*

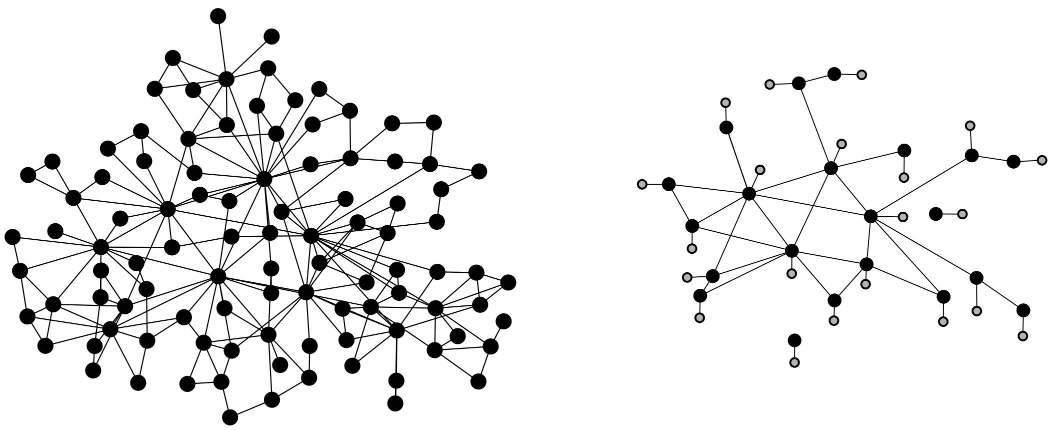

Figure 2.3: The Figure illustrates an example of reducing a (planar) graph with respect to Rule 1 and Rule 2. The original graph on the left-hand side is reduced to the graph on the right-hand side. The little grey vertices of degree one in the reduced graph are the "gadget vertices" that were introduced by the reduction rules to force their neighbors to be in an optimal dominating set. Since all 21 remaining non-gadget vertices have such a degree-one neighbor, we already obtain an optimal dominating set (which consists of these 21 non-gadget vertices).

Proof: The proof is deferred to the Appendix at the end of the chapter. □

We remark that the running times given in Lemmas 2.2.4 and 2.2.6 are pure worst-case estimates and turn out to be much lower in our experimental studies (see Subsection 6.4). In particular, for practical purposes it is important to see that Rule 2 can only be applied for vertex pairs that are at distance at most three.

2.2.1.3 Reduced Graphs

We will now describe the problem kernel reduction for the par-DOMINATING SET problem.

Definition 2.2.7 *Let* $G = (V, E)$ *be a graph such that both the application of Rule 1 and the application of Rule 2 leave the graph unchanged. Then we say that* G *is* reduced *with respect to these rules.*

Given an input instance (G, k) of the par-DOMINATING SET problem, we can repeatedly apply Rule 1 and Rule 2 to the graph G to obtain a reduced graph G'. An example illustrating the transformation of G into G' is demonstrated in Fig. 2.3. The problem kernel reduction then is given by $\Phi : (G, k) \mapsto (G', k)$.

Observing that the (successful) application of any reduction rule always "shrinks" the given graph implies that there can only be $O(n)$ successful applications of reduction rules. This leads to the following.

Lemma 2.2.8 *A graph* G *can be transformed into a reduced graph* G' *with* $\mathrm{ds}(G) = \mathrm{ds}(G')$ *in time* $O(n^3)$ *in the planar case and in time* $O(n^5)$ *in the general case.* □

Remark 2.2.9 The reduction of a graph G to a reduced graph G' does not only guarantee that $\mathrm{ds}(G) = \mathrm{ds}(G')$. Moreover, the reduction is constructive, in the sense that an optimal dominating set D' for $G' = (V', E')$ easily translates into an optimal dominating set D for $G = (V, E)$: By construction, $V' = V_{\text{gadget}} \cup V_0$, where $V_0 \subseteq V$ and V_{gadget} are some newly added "gadget vertices," which are either single vertices of degree one or a pair of "gadget vertices" of degree two (as added by Case (1.1) of Rule 2). If $D' \subseteq V'$ is a dominating set for G', then—noting that a single vertex of a "gadget" pair cannot solely be part of D— it is easy to see that $D := (D' \cap V_0) \cup N(D' \cap V_{\text{gadget}})$ is an optimal dominating set for G. It is, however, not possible to reconstruct *all* optimal dominating sets for G from G'.

The following remark is crucial for the proof of the linear problem kernel.

Remark 2.2.10 A graph $G = (V, E)$ which is reduced with respect to reduction Rules 1 and 2 has the following properties:

(i) For all $v \in V$, the set $N_{\text{prison}}(v)$ of prisoner vertices is empty, since otherwise Rule 1 removes these vertices. There is one exception for this statement: A newly added "gadget vertex" v' of v formally belongs to $N_{\text{prison}}(v)$. But except for such a single gadget vertex, the set $N_{\text{prison}}(v)$ is empty.

(ii) For all $v, w \in V$, there exists a single vertex in $N_{\text{guard}}(v, w) \cup N_{\text{prison}}(v, w)$ which dominates all prisoner vertices $N_{\text{prison}}(v, w)$, since otherwise Rule 2 removes these vertices.

2.2.2 Region Decompositions

In the following, we restrict to planar graphs. The key result will be the following.

Proposition 2.2.11 *For a planar graph* $G = (V, E)$ *which is reduced with respect to reduction Rules 1 and 2, we get* $|V| \leq 335\ \mathrm{ds}(G)$.

The rest of this section is devoted to the proof of Proposition 2.2.11. Together with Lemma 2.2.8 this will establish Theorem 2.2.1.

History and outline of the proof. The linear size problem kernel question for par-DOMINATING SET on planar graphs has taken the attention of numerous people. Various proof strategies were suggested, however, none of them yielded a complete, correct proof. We briefly comment on some previous attempts showing partial results and outline where the obviously hard part of the proof lies.

Suppose we are given a reduced planar graph $G = (V, E)$ together with a dominating set D with $|D| = \mathrm{ds}(G)$. The task is to prove an upper bound of the form $|V| \leq c \cdot \mathrm{ds}(G)$ for some constant c.

(i) A first idea that we followed for a long time is to partition the vertices in $V \setminus D$ into disjoint subsets $V_i := \{v \in V \setminus D \mid |N(v) \cap D| = i\}$ according to the number of neighbors in D. We wish to give upper bounds for each of these sets individually. For the size of $V_{\geq 3} := \bigcup_{i \geq 3} V_i$, an upper bound of the form $3\,\mathrm{ds}(G) - 4$ is quite easy to see, by analyzing the planar bipartite graph $G[D \cup V_{\geq 3}]$. For the set V_2, we might consider the graph $G_{V_2} := (D, E_{V_2})$ with $E_{V_2} := \{\{x,y\} \mid \exists v \in V_2 : \{x,v\},\{v,y\} \in E\}$, where the vertices of V_2 are interpreted as edges between vertices in D. This is a planar graph with possible multiple edges. Since the graph is reduced, one can show that there are not too many of these multiple edges and that $|V_2| = |E_{V_2}| = O(\mathrm{ds}(G))$.[3] Hence, it would remain to show that $|V_1| = O(\mathrm{ds}(G))$. This, however, seemed to be a difficult task, for which similar tricks as used for the upper bounds of V_2 and $V_{\geq 3}$ did not work.

(ii) An alternative proof strategy could be to try to upperbound the number of vertices which lie in a neighborhood $N(v, w)$ of a pair $v, w \in D$. An attempt that we followed was to find (at most $O(\mathrm{ds}(G))$ many) neighborhoods $N(v_1, w_1), \ldots, N(v_\ell, w_\ell)$ with $v_i, w_i \in D$, such that all vertices in V lie in at least one such neighborhood; and then use the fact that G is reduced in order to prove that each $N(v_i, w_i)$ has size at most $O(1)$. However, even if the graph G is reduced, the neighborhoods $N(v, w)$ for vertices $v, w \in D$ may contain many vertices: the size of $N(v, w)$, basically depends on how big $N_{\mathrm{exit}}(v, w)$ is.

(iii) The strategy by which we finally succeeded, and which will be outlined in the rest of this chapter, circumvents these difficulties. Instead of trying to partition the vertices in $O(\mathrm{ds}(G))$ many neighborhoods of the form $N(v_i, w_i)$ (for some $v_i, w_i \in D$), our partition sets will be so-called "regions," which are only those vertices of a neighborhood $N(v_i, w_i)$, that lie on *short* paths between v_i and w_i. By this we ensure that only a constant number of exit vertices from $N_{\mathrm{exit}}(v_i, w_i)$ are contained in a region. This is the key for showing that the size of such a region is bounded by $O(1)$ (the fact that G is reduced with respect to Rule 2 is important in this step). The drawback of this method is that not all vertices in $V \setminus D$ lie in such regions. However, if a maximal number of regions is used, i.e., if we use a so-called "maximal region-decomposition," only $O(\mathrm{ds}(G))$ vertices will lie outside such regions (the fact that G is reduced with respect to Rule 1 being important here).

The notion of "region-decompositions" heavily relies on the planarity of our input graph and cannot be carried over to general graphs.

We now give the details for this last strategy. In the remainder of this section let us consider some fixed embedding ϕ of $G = (V, E)$ in the plane.[4]

Definition 2.2.12 *Let* $G = (V, E)$ *be a plane graph. A* region $R(v, w)$ *between two vertices* v, w *is a closed subset of the plane with the following properties:*

[3] The trick is to show that every multiple edge between $x, y \in D$ corresponds to a set of so-called simple regions (see Definition 2.2.19) between x and y. Moreover, the induced graph of these simple regions (see Definition 2.2.14) is "thin" in the sense of Definition 2.2.15.

[4] For the ease of presentation, since it will always be clear from the context, in the following, we will not distinguish between a vertex $v \in V$ and the point $\phi(v)$ in the plane, or an edge $e \in E$ and the arc $\phi(e)$. Moreover, we simply write G for the plane graph (G, ϕ).

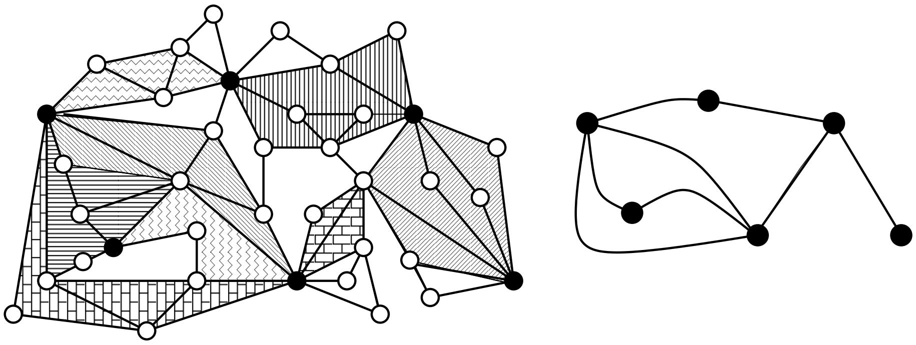

Figure 2.4: The left-hand side diagram shows an example of a possible D-region decomposition $\mathcal{R}$ of some graph G, where D is the subset of vertices in G that are drawn in black (D is a dominating set here). The various regions are highlighted by different patterns. The remaining white areas are not considered as regions. The given D-region decomposition is maximal since we cannot find a further region between vertices in D that does not cross any region in $\mathcal{R}$ and contains a vertex from $V \setminus V(\mathcal{R})$. The right-hand side shows the induced graph $G_{\mathcal{R}}$ (Definition 2.2.14).

(i) *the boundary of* $R(v, w)$ *is formed by two paths* P_1 *and* P_2 *in* V *which connect* v *and* w*, and the length of each path is at most three*[5]*, and*

(ii) *all vertices which are strictly inside the region* $R(v, w)$ *are from* $N(v, w)$.[6]

For a region $R = R(v, w)$*, let* $V(R)$ *denote the vertices belonging to* R*, i.e.,*

$$V(R) := \{u \in V \mid u \textit{ sits inside or on the boundary of } R\}.$$

Definition 2.2.13 *Let* $G = (V, E)$ *be a plane graph and* $D \subseteq V$*. A* D-region decomposition *of* G *is a set* $\mathcal{R}$ *of regions between vertices in* D *such that*

(i) *for* $R(v, w) \in \mathcal{R}$*, no vertex from* D *(except for* v, w*) lies in* $V(R(v, w))$*, and*

(ii) *no two regions* $R_1, R_2 \in \mathcal{R}$ *intersect (however, they may touch each other by having common boundaries).*

For a D*-region decomposition* $\mathcal{R}$*, we define* $V(\mathcal{R}) := \bigcup_{R \in \mathcal{R}} V(R)$*. A* D*-region decomposition* $\mathcal{R}$ *is called* maximal *if there is no region* $R \notin \mathcal{R}$ *such that* $\mathcal{R}' := \mathcal{R} \cup \{R\}$ *is a* D*-region decomposition with* $V(\mathcal{R}) \subsetneq V(\mathcal{R}')$.

For an example of a (maximal) D-region decomposition we refer to the left-hand side diagram of Fig. 2.4.

We will show that, for a given graph G with dominating set D, we can always find a maximal D-region decomposition with at most $O(\mathrm{ds}(G))$ many regions. For that purpose, we observe that a D-region decomposition induces a graph in a very natural way.

[5]The length of a path is the number of edges on it.

[6]By "strictly inside the region $R(v, w)$" we mean lying in the region, but not sitting on the boundary of $R(v, w)$.

Definition 2.2.14 *The* induced graph $G_{\mathcal{R}} = (V_{\mathcal{R}}, E_{\mathcal{R}})$ *of a* D*-region decomposition* $\mathcal{R}$ *of* G *is the graph with possible multiple edges which is defined as follows:*

$$V_{\mathcal{R}} := D, \text{ and } E_{\mathcal{R}} := \{\{v, w\} \mid \text{there is a region } R(v, w) \in \mathcal{R} \text{ between } v, w \in D\}.$$

Note that, by Definition 2.2.13, the induced graph $G_{\mathcal{R}}$ of a D-region decomposition is planar. For an example of an induced graph $G_{\mathcal{R}}$, see Fig. 2.4.

Definition 2.2.15 *A planar graph* $G = (V, E)$ *with multiple edges is* thin *if there exists a planar embedding such that no two multiple edges are homotopic: This means that if there are two edges* e_1, e_2 *between a pair of distinct vertices* $v, w \in V$*, then there must be two further vertices* $u_1, u_2 \in V$ *which sit inside the two disjoint areas of the plane that are enclosed by* e_1, e_2*.*

The induced graph $G_{\mathcal{R}}$ in Fig. 2.4 is thin.

Lemma 2.2.16 *For a thin planar graph* $G = (V, E)$ *we have* $|E| \leq 3|V| - 6$*.*

Proof (Sketch): The claim is true for planar graphs without multiple edges. But then, an easy induction on the number of multiple edges in G proves the claim. □

Using the notion of thin graphs, we can formulate the main result of this section.

Proposition 2.2.17 *For a reduced plane graph* G *with dominating set* D*, there exists a maximal* D*-region decomposition* $\mathcal{R}$ *such that* $G_{\mathcal{R}}$ *is thin.*

Proof: We give a constructive proof on how to find a maximal D-region decomposition $\mathcal{R}$ of a plane graph G such that the induced graph $G_{\mathcal{R}}$ is thin. Consider the algorithm presented in Fig. 2.5. It is obvious that this algorithm returns a D-region decomposition, since—by construction—we made sure that all regions in $\mathcal{R}$ are between vertices in D, that regions in $\mathcal{R}$ do not contain vertices from D, and that regions in $\mathcal{R}$ do not intersect mutually. Moreover, the D-region decomposition obtained by the algorithm is maximal: If a vertex u does not belong to a region, i.e., if $u \notin V_{\text{used}}$, then eventually we check, if there is a region S_u such that $\mathcal{R} \cup \{S_u\}$ is a D-region decomposition.

It remains to show that the induced graph $G_{\mathcal{R}}$ of the D-region decomposition $\mathcal{R}$ found by the algorithm is thin. We embed $G_{\mathcal{R}}$ in the plane in such a way that an edge belonging to a region $R \in \mathcal{R}$ is drawn inside the area covered by R. To see that the graph is thin, we have to show that, for every multiple edge e_1, e_2 (belonging to two regions $R_1, R_2 \in \mathcal{R}$ that were chosen at some point of the algorithm) between two vertices $v, w \in D$, there exist two vertices $u_1, u_2 \in D$ which lie inside the areas enclosed by e_1, e_2. Let A be such an area. Suppose that there is no vertex $u \in D$ in A. We distinguish two cases. Either there is also no vertex from $V \setminus D$ in A, or there are other vertices V' from $V \setminus D$ inside A. In the first case, by joining the regions R_1 and R_2 we obtain a bigger region which fulfills all the four conditions (i)–(iv) checked by the algorithm in Fig. 2.5, a contradiction to the maximality of R_1 and R_2. In the second case, since D is assumed to be a dominating set, the vertices in V' need to be dominated by D. Since v, w are the only vertices from D which are part of A, R_1, or R_2, the vertices in V' need to be dominated

```
procedure region_decomposition
/* input:   a plane graph G = (V, E) and a vertex subset D ⊆ V.  */
/* output:  a D-region decomposition 𝓡 for G                      */
/*          such that the induced graph G_𝓡 is thin.               */
```

- $V_{\text{used}} \leftarrow \emptyset$, $\mathcal{R} \leftarrow \emptyset$.
- `for all` $u \in V$ `do`
  - `if` (($u \notin V_{\text{used}}$) `and` ($u \in V(R)$ for some region $R = R(v, w)$ between two vertices $v, w \in D$ such that $\mathcal{R} \cup \{R\}$ is a D-region decomposition)) `then`
    - consider the set $\mathcal{R}_u$ of all regions S with the following properties:[a]
      - (i) S is a region between v and w.
      - (ii) S contains u.
      - (iii) no vertex from $D \setminus \{v, w\}$ is in $V(S)$.
      - (iv) S does not cross any region from $\mathcal{R}$.
    - choose a region $S_u \in \mathcal{R}_u$ which is maximal in space.[b]
    - $\mathcal{R} \leftarrow \mathcal{R} \cup \{S_u\}$.
    - $V_{\text{used}} \leftarrow V_{\text{used}} \cup V(S_u)$.
- `return` $\mathcal{R}$.

[a]These four properties ensure that $\mathcal{R} \cup \{S\}$ is a D-region decomposition for every $S \in \mathcal{R}_u$.
[b]By "maximal in space" we mean a region S_u such that $S' \supseteq S_u$ for any $S' \in \mathcal{R}_u$ implies $S' = S_u$.

Figure 2.5: Greedy-like construction of a maximal D-region decomposition (see Proposition 2.2.17).

by v, w, hence they belong to $N(v, w)$. But then again by joining the regions R_1 and R_2 we obtain a bigger region which again fulfills all the four conditions (i)–(iv) of the algorithm in Fig. 2.5, a contradiction to the maximality of R_1 and R_2. □

2.2.3 An Upper Bound on the Kernel Size

Suppose that we are given a reduced planar graph $G = (V, E)$ with a minimum dominating set D. Then, by Proposition 2.2.17 and Lemma 2.2.16, we find a maximal D-region decomposition $\mathcal{R}$ of G with at most $O(\text{ds}(G))$ regions. To see that $|V| = O(\text{ds}(G))$, it remains to show that

(i) there are at most $O(\text{ds}(G))$ vertices not belonging to any of the regions in $\mathcal{R}$, and that

(ii) every region of $\mathcal{R}$ contains at most $O(1)$ vertices.

These issues are treated by Propositions 2.2.21 and 2.2.22.

We first of all state two technical lemmas, one which characterizes an important property of a maximal region decomposition and another one which gives an upper bound on the size of a special type of a region.

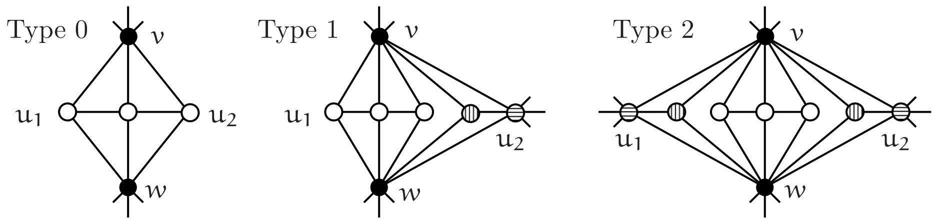

Figure 2.6: Simple regions of Type 0, Type 1, and Type 2. This figure illustrates the largest possible simple regions in a reduced graph. Vertices marked with horizontal lines are in $N_{\text{exit}}(v, w)$, vertices marked with vertical lines belong to $N_{\text{guard}}(v, w)$, and white vertices are in $N_{\text{prison}}(v, w)$.

Lemma 2.2.18 *Let* G *be a reduced planar graph with a dominating set* D *and let* $\mathcal{R}$ *be a maximal* D*-region decomposition. If* $u \in N_{exit}(v)$ *for some vertex* $v \in D$ *then* $u \in V(\mathcal{R})$*.*

Proof: The proof is deferred to the Appendix at the end of the chapter. □

We now investigate a special type of a region specified by the following definition.

Definition 2.2.19 *A region* $R(v, w)$ *between two vertices* $v, w \in D$ *is called* simple *if all vertices contained in* $R(v, w)$ *except for* v, w *are common neighbors of both* v *and* w*, i.e., if* $(V(R(v, w)) \setminus \{v, w\}) \subseteq N(v) \cap N(w)$*.*

Let v, u_1, w, u_2 *be the vertices that sit on the boundary of the simple region* $R(v, w)$*. We say that* $R(v, w)$ *is a simple region of* Type i $(0 \leq i \leq 2)$ *if* i *vertices from* $\{u_1, u_2\}$ *have a neighbor outside* $R(v, w)$*.*

Lemma 2.2.20 *Every simple region* R *of Type* i *of a plane reduced graph contains at most* $5+2i$ *vertices.*

Proof: The proof is deferred to the Appendix at the end of the chapter. □

Fig. 2.6 illustrates the (worst-case) examples for simple regions of each type. We use Lemmas 2.2.18 and 2.2.20 for the following two proofs.

Proposition 2.2.21 *Let* $G = (V, E)$ *be a plane reduced graph and let* D *be a dominating set of* G*. If* $\mathcal{R}$ *is a maximal* D*-region decomposition, then* $|V \setminus V(\mathcal{R})| \leq 2|D| + 56|\mathcal{R}|$*.*

Proof: To start with, we claim that every vertex $u \in V \setminus V(\mathcal{R})$ is either a vertex in D or belongs to a set $N_{\text{guard}}(v) \cup N_{\text{prison}}(v)$ for some $v \in D$. To see this, suppose that $u \notin D$. But since D is a dominating set, we know that $u \in N(v) = N_{\text{exit}}(v) \cup N_{\text{guard}}(v) \cup N_{\text{prison}}(v)$ for some vertex $v \in D$. Since $\mathcal{R}$ is assumed to be maximal, by Lemma 2.2.18, we know that $N_{\text{exit}}(v) \subseteq V(\mathcal{R})$. Thus, $u \in N_{\text{guard}}(v) \cup N_{\text{prison}}(v)$.

For a vertex $v \in D$, let $N^*_{\text{guard}}(v) = N_{\text{guard}}(v) \setminus V(\mathcal{R})$. The above observation implies

$$V \setminus V(\mathcal{R}) \subseteq D \cup \Big(\bigcup_{v \in D} N_{\text{prison}}(v)\Big) \cup \Big(\bigcup_{v \in D} N^*_{\text{guard}}(v)\Big). \tag{2.1}$$

From this set inclusion, we will derive an upper bound on $|V \setminus V(\mathcal{R})|$ in what follows.

We, firstly, upperbound the size of $\bigcup_{v \in D} N_{\text{prison}}(v)$. Since G is reduced, by Remark 2.2.10, $|N_{\text{prison}}(v)| \leq 1$, we get $|\bigcup_{v \in D} N_{\text{prison}}(v) \leq |D|$.

We now upperbound the size of $N^*_{\text{guard}}(v)$ for a given vertex $v \in D$. To this end, for a vertex $v \in D$, let $N^*_{\text{exit}}(v)$ be the subset of $N_{\text{exit}}(v)$ which sit on the boundary of a region in $\mathcal{R}$. It is clear that $N^*_{\text{guard}}(v) \subseteq N(v) \cap N(N^*_{\text{exit}}(v))$. Hence, we investigate the set $N^*_{\text{exit}}(v)$. Suppose that $R(v, w_1), \ldots, R(v, w_\ell)$ are the regions between v and some other vertices $w_i \in D$, where $\ell = \deg_{G_{\mathcal{R}}}(v)$ is the degree of v in the induced region graph $G_{\mathcal{R}}$. Then, every region $R(v, w_i)$ can contribute at most two vertices u_i^1, u_i^2 to $N^*_{\text{exit}}(v)$, i.e., in the worst-case, we have $N^*_{\text{exit}}(v) = \bigcup_{i=1}^{\ell} \{u_i^1, u_i^2\}$ with $u_i^1, u_i^2 \in V(R(v, w_i))$, i.e., $|N^*_{\text{exit}}(v)| \leq 2 \deg_{G_{\mathcal{R}}}(v)$. We already observed that every vertex in $N^*_{\text{guard}}(v)$ must be a common neighbor of v and some vertex in $N^*_{\text{exit}}(v)$. We claim that, moreover, the vertices in $N^*_{\text{guard}}(v)$ can be grouped into various simple regions. More precisely, we claim that there exists a set $\mathcal{S}_v$ of simple regions, such that

(i) every $S \in \mathcal{S}_v$ is a simple region between v and some vertex in $N^*_{\text{exit}}(v)$,

(ii) $N^*_{\text{guard}}(v) \subseteq \bigcup_{S \in \mathcal{S}_v} V(S)$, and

(iii) $|\mathcal{S}_v| \leq 2 \cdot |N^*_{\text{exit}}(v)|$.

The idea for the construction of the set $\mathcal{S}_v$ is similar to the greedy-like construction of a maximal region decomposition (see Fig. 2.5). Starting with $\mathcal{S}_v$ as empty set, one iteratively adds a *simple* region $S(v, x)$ between v and some vertex $x \in N^*_{\text{exit}}(v)$ to the set $\mathcal{S}_v$ in such a way that (1) $\mathcal{S}_v \cup \{S(v, x)\}$ contains more $N^*_{\text{guard}}(v)$-vertices than $\mathcal{S}_v$, (2) $S(v, x)$ does not cross any region in $\mathcal{S}_v$, and (3) $S(v, x)$ is maximal (in space) under all simple regions S between v and x that do not cross any region in $\mathcal{S}_v$. The fact that we end up with at most $2 \cdot |N^*_{\text{exit}}(v)|$ many regions can be proven by an argument on the induced graph $G_{\mathcal{S}_v}$.

Since, by Lemma 2.2.20, every simple region $S(v, x)$ with $x \in N^*_{\text{exit}}(v)$ contains at most seven vertices—not counting the vertices v and x which clearly cannot be in $N^*_{\text{guard}}(v)$—we conclude from (i)–(iii) that

$$|N^*_{\text{guard}}(v)| \leq 7 \cdot |\mathcal{S}_v| \leq 14 \cdot |N^*_{\text{exit}}(v)| \leq 28 \cdot \deg_{G_{\mathcal{R}}}(v).$$

From the set inclusion in (2.1) we then get

$$|V \setminus V(\mathcal{R})| \;\leq\; |D| + |D| + \sum_{v \in D} |N^*_{\text{guard}}(v)| \;\leq\; 2 \cdot |D| + 28 \sum_{v \in D} \deg_{G_{\mathcal{R}}}(v) \;\leq\; 2 \cdot |D| + 56 \cdot |\mathcal{R}|.$$

□

We now investigate the maximal size of a region in a reduced graph. The worst-case scenario for a region in a reduced graph is depicted in Fig. 2.7.

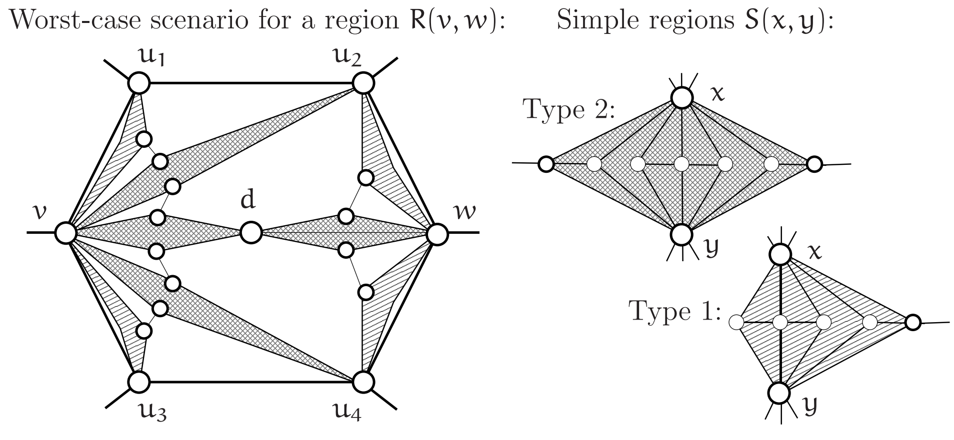

Figure 2.7: The left-hand diagram shows a worst-case scenario (with 55 vertices) for a region $R(v,w)$ between two vertices v and w in a reduced planar graph (cf. the proof of Proposition 2.2.22). Such a region may contain up to four vertices from $N_{\text{exit}}(v,w)$, namely u_1, u_2, u_3, and u_4. The vertices from $R(v,w)$ which belong to the sets $N_{\text{guard}}(v,w)$ and $N_{\text{prison}}(v,w)$ can be grouped into so-called simple regions of Type 1 (marked with a line-pattern) or of Type 2 (marked with a crossing-pattern); the structure of such simple regions $S(x,y)$ is given in the right-hand part of the diagram. In $R(v,w)$ there might be two simple regions $S(d,v)$ and $S(d,w)$ (of Type 2), containing vertices from $N_{\text{prison}}(v,w)$. And, we can have up to six simple regions of vertices from $N_{\text{guard}}(v,w)$: $S(u_1,v), S(v,u_3), S(u_4,w), S(w,u_2), S(u_2,v)$, and $S(u_4,v)$ (among these, the latter two can be of Type 2 and the others are of Type 1). See the proof of Proposition 2.2.22 for details.

Proposition 2.2.22 *A region* R *of a plane reduced graph contains at most* 55 *vertices, i.e.,* $|V(R)| \leq 55$.

Proof: Let $R = R(v,w)$ be a region between vertices $v, w \in V$. As in the proof of Lemma 2.2.20, we count the number of vertices in $V(R) \subseteq N[v,w] = N[v] \cup N[w]$ which belong to $N_{\text{exit}}(v,w)$, $N_{\text{guard}}(v,w)$, and $N_{\text{prison}}(v,w)$, separately.

We start with counting the number of vertices in $N_{\text{prison}}(v,w) \cap V(R)$. Since the graph is assumed to be reduced, by Remark 2.2.10, we know that all vertices in $N_{\text{prison}}(v,w)$ need to be dominated by a single vertex from $N_{\text{guard}}(v,w) \cup N_{\text{prison}}(v,w)$. Denote by d the vertex which dominates all vertices in $N_{\text{prison}}(v,w)$. Since all vertices in $N_{\text{prison}}(v,w)$ are also dominated by v or w, we may write $N_{\text{prison}}(v,w) = S(d,v) \cup S(d,w)$ where $S(d,v) \subseteq N(d) \cap N(v)$ and $S(d,w) \subseteq N(d) \cap N(w)$. In this way, $S(d,v)$ and $S(d,w)$ form simple regions between d and v, and d and w, respectively. In Fig. 2.7 these two simple regions $S(d,v)$ and $S(d,w)$ (of Type 2) are horizontally arranged inside R and they are drawn with a crossing pattern. By Lemma 2.2.20 we know that $S(d,v)$ and $S(d,w)$ both contain at most seven vertices each, not counting the vertices d, v and d, w, respectively. Since d may be from $N_{\text{prison}}(v,w)$, we obtain $|N_{\text{prison}}(v,w) \cap V(R)| \leq 2 \cdot 7 + 1 = 15$.

It is clear that vertices in $N_{\text{exit}}(v,w) \cap V(R)$ need to be on the boundary of R, since, by definition of $N_{\text{exit}}(v,w)$, they have a neighbor outside $N(v,w)$. The region R is enclosed by two paths P_1 and P_2 between v and w of length at most three each. Hence, there can be at most four vertices in $N_{\text{exit}}(v,w) \cap V(R)$, where this worst-case holds if P_1 and P_2 are disjoint and have length exactly three each. Consider Fig. 2.7, which shows a region enclosed by two such paths. Suppose that the four vertices on the boundary besides v and w are u_1, u_2, u_3, and u_4.

Finally, we count the number of vertices in $N_{\text{guard}}(v,w) \cap V(R)$. It is important to note that, by definition of $N_{\text{guard}}(v,w)$, every such vertex needs to have a neighbor in $N_{\text{exit}}(v,w)$ and at the same time needs to be a neighbor of either v or w (or both). Hence, $N_{\text{guard}}(v,w) = \bigcup_{i=1}^{4}(S(u_i,v) \cup S(u_i,w))$, where $S(u_i,v) \subseteq N(u_i) \cap N(v)$ and $S(u_i,w) \subseteq N(u_i) \cap N(w)$. All the sets $S(u_i,v)$ and $S(u_i,w)$, where $1 \leq i \leq 4$, form simple regions inside R. Due to planarity, however, there cannot exist all eight of these regions. In fact, in order to avoid crossings, the worst-case scenario is depicted in Fig. 2.7 where six of these simple regions exist.[7] Concerning the type of these simple regions, it is not hard to verify, that in the worst-case there can be two among these six regions of Type 2, the other four of them being of Type 1. In Fig. 2.7, the simple regions $S(u_2,v)$ and $S(u_4,v)$ are of Type 2 (having two connections to vertices outside the simple region), and the simple regions $S(u_1,v), S(u_2,w), S(u_3,v)$, and $S(u_4,w)$ are of Type 1 (having only one connection to vertices outside the region; a second connection to vertices outside the region is not possible because of the edges $\{u_1,v\},\{u_2,w\},\{u_3,v\}$, and $\{u_4,w\}$). In summary, the worst-case number of vertices in $N_{\text{guard}}(v,w) \cap V(R)$ is given by four times the number of vertices of a simple region of Type 1 and two times the number of vertices of a simple region of Type 2; each time, of course, excluding vertices from $\{u_1,u_2,u_3,u_4,v,w\}$. By Lemma 2.2.20 this amounts to $|N_{\text{guard}}(v,w) \cap V(R)| \leq 4 \cdot (3+2\cdot 1) + 2 \cdot (3 + 2 \cdot 2) = 34$.[8]

The claim now follows from the fact that $V(R) = \{v,w\} \cup (V(R) \cap N_{\text{prison}}(v,w)) \cup (V(R) \cap N_{\text{exit}}(v,w)) \cup (V(R) \cap N_{\text{guard}}(v,w))$, which yields $|V(R)| = 2 + 15 + 4 + 34 = 55$. □

We now piece all arguments together to give a complete proof of Proposition 2.2.11. We first of all observe that, for a graph G with minimum dominating set D, by Proposition 2.2.17 and Lemma 2.2.16, we can find a D-region decomposition $\mathcal{R}$ of G with at most $3\,\text{ds}(G)$ regions, i.e., $|\mathcal{R}| \leq 3\,\text{ds}(G)$. By Proposition 2.2.22, we know that

$$|V(\mathcal{R})| \;\leq\; \sum_{R \in \mathcal{R}} |V(R)| \;\leq\; 55|\mathcal{R}|.$$

By Proposition 2.2.21, we have $|V \setminus V(\mathcal{R})| \leq 2|D| + 56|\mathcal{R}|$. Hence, we get

$$|V| \;\leq\; 2|D| + 111|\mathcal{R}| \;\leq\; 335\ \text{ds}(G).$$

This finishes the proof of Proposition 2.2.11.

[7] Observe that regions $S(u_1,w)$ and $S(u_3,w)$ would cross the regions $S(u_2,v)$ and $S(u_4,v)$, respectively.

[8] Note that for the size of, e.g., a region $S(u_i,v)$ we do not have to count u_i and v, since they are not vertices in $N_{\text{guard}}(v,w)$.

Appendix

Proof of Lemma 2.2.4:
We first of all discuss the planar case. To carry out Rule 1, for each vertex v of the given planar graph G we first have to determine the neighbor sets $N_{\text{exit}}(v)$, $N_{\text{guard}}(v)$, and $N_{\text{prison}}(v)$. By definition of these sets, one easily observes that it is sufficient to consider the subgraph G that is induced by all vertices that are connected to v by a path of length at most two. To do so, we employ a search tree of depth two, rooted at v. We perform two phases.

In phase 1, constructing the search tree we determine the vertices from $N_{\text{exit}}(v)$. Each vertex of the first level (i.e., at distance one from the root v) of the search tree that has a neighbor at the second level of the search tree belongs to $N_{\text{exit}}(v)$. Observe that it is enough to stop the expansion of a vertex from the first level as soon as its *first* neighbor in the second level is encountered. Hence, denoting the degree of v by $\deg(v)$, phase 1 takes time $O(\deg(v))$, because there clearly are at most $2 \cdot \deg(v)$ tree edges and at most $O(\deg(v))$ non-tree edges to be explored. The latter holds true since these non-tree edges all belong to the subgraph of G induced by $N[v]$. Since this graph is clearly planar and $|N[v]| = \deg(v) + 1$, the claim follows.

In phase 2, it remains to determine the sets $N_{\text{guard}}(v)$ and $N_{\text{prison}}(v)$. To get $N_{\text{guard}}(v)$, one basically has to go through all vertices from the first level of the above search tree that are not already marked as being in $N_{\text{exit}}(v)$, but have at least one neighbor in $N_{\text{exit}}(v)$. All this can be done within the planar graph induced by $N[v]$, using the already marked $N_{\text{exit}}(v)$-vertices, in time $O(\deg(v))$. Finally, $N_{\text{prison}}(v)$ simply consists of vertices from the first level that are neither marked being in $N_{\text{exit}}(v)$ nor marked being in $N_{\text{guard}}(v)$. In summary, this shows that for vertex v, the sets $N_{\text{exit}}(v)$, $N_{\text{guard}}(v)$, and $N_{\text{prison}}(v)$ can be constructed in time $O(\deg(v))$.

Once having determined these three sets, the sizes of which all are bounded by $\deg(v)$, it is clear that the possible removal of vertices from $N_{\text{guard}}(v)$ and $N_{\text{prison}}(v)$ and the addition of a vertex and an edge as required by Rule 1 all can be done in time $O(\deg(v))$. Finally, it remains to analyze the overall complexity of this procedure when going through all n vertices of $G = (V, E)$. But this is easy. The running time can be bounded by $\sum_{v \in V} O(\deg(v))$. Since G is planar, this sum is bounded by $O(n)$, i.e., the whole reduction takes linear time.

For general graphs, the method described above leads to a worst-case cubic time implementation of Rule 1. Here, one ends up with the sum $\sum_{v \in V} O((\deg(v))^2) = O(n^3)$. □

Proof of Lemma 2.2.6:
To prove the time bounds for Rule 2, basically the same ideas as for Rule 1 apply (cf. proof of Lemma 2.2.4). Instead of a depth two search tree, one now has to argue on a search tree where the levels indicate the minimum of the distances to vertex v and w. Hence, we associate the vertices v and w to the root of this search tree. The first level consists of all vertices that lie in $N(v, w)$ (i.e., at distance one from either of the vertices v or w). Determining the subset $N_{\text{prison}}(v, w)$ means to check whether some vertex on the first level has a neighbor on the second level. We do the same kind of construction as in Lemma 2.2.4. The running time again is determined by the size of the subgraph induces by the vertices that correspond to the root and the first level of this search tree, i.e., by $G[N[v, w]]$ in this case. For planar graphs, we have $|G[N[v, w]]| = O(\deg(v) + \deg(w))$. Hence, we get $\sum_{v,w \in V} O(\deg(v) + \deg(w))$ as an

upper bound on the overall running time in the case of planar graphs. This is upperbounded by

$$O(\sum_{v \in V}(n \cdot \deg(v) + \sum_{w \in V} \deg(w))) = O(n^2).$$

Since, in case of general graphs, we have $|G[N[v,w]]| = O\left((\deg(v) + \deg(w))^2\right)$, we trivially obtain the upper bound $\sum_{v,w \in V} O((\deg(v) + \deg(w))^2) = O(n^4)$ for the overall running time. □

Proof of Lemma 2.2.18:
Let $u \in N_{\text{exit}}(v)$ for some $v \in D$ and assume that $u \notin V(\mathcal{R})$. By definition of $N_{\text{exit}}(v)$, there exists a vertex $u' \in N(u)$ with $u' \notin N[v]$. We distinguish two cases. Either $u' \in D$ or u' needs to be dominated by a vertex $w \in D$ with $w \neq v$. If $u' \in D$, we consider the (degenerated) region consisting of the path $\langle v, u, u' \rangle$. Since $\mathcal{R}$ is assumed to be maximal, this path must cross a region $R \in \mathcal{R}$. But this implies that $u \in V(R)$, a contradiction.

In the second case (i.e., if u' is dominated by some other vertex $w \neq v$), we consider the (degenerated) region consisting of the path $\langle v, u, u', w \rangle$. Again, by maximality of $\mathcal{R}$, this path must cross a region $R = R(x, y) \in \mathcal{R}$ between two vertices $x, y \in D$. Since, by assumption, $u \notin V(R)$, the edge $\{u', w\}$ has to cross R which implies that w lies on the boundary of or inside R and, hence, $w \in V(R)$. However, according to the definition of a D-region decomposition, the only vertices from D that are in $V(R)$ are x, y. Hence, w.l.o.g., $x = w$. At the same time u' must lie on the boundary of R, otherwise $u \in V(R)$. By definition of a region, there exists a path P of length at most three between w and y that goes through u' and that is part of the boundary of R. We claim that u' is a neighbor of y: If this were not the case, the edge $\{u', w\}$ would be *on* P. We already remarked, however, that the edge $\{u', w\}$ *crosses* R and, thus, cannot lie on the boundary, a contradiction to u' not being neighbor of y. We know that $u' \notin N(v)$, hence, $y \neq v$. But then, the (degenerated) region R' consisting of the path $\{v, u, u', y\}$ is a region between two vertices v and y in D, which does not cross (it only touches R) any region in $\mathcal{R}$. For the D-region decomposition $\mathcal{R}' := \mathcal{R} \cup \{R'\}$, we have $u \in V(\mathcal{R}') \setminus V(\mathcal{R})$, contradicting the maximality of $\mathcal{R}$. □

Proof of Lemma 2.2.20:
Let $R = R(v, w)$ be a simple region of Type i between vertices v and w. We will show that $|V(R)| \leq 5 + 2i$. The worst-case simple regions are depicted in Fig. 2.6. Firstly, let us count the number of vertices in $V(R)$ which belong to $N_{\text{exit}}(v, w) \cup N_{\text{guard}}(v, w)$. Clearly, only vertices on the boundary (except for v and w) can have a neighbor outside R. Thus, all vertices in $N_{\text{exit}}(v, w) \cap V(R)$ lie on the boundary of R. By definition of a simple region of Type i, we have $|N_{\text{exit}}(v, w) \cap V(R)| \leq i$. Moreover, it is easy to see that, by planarity, every vertex in $N_{\text{exit}}(v, w) \cap V(R)$ can contribute at most one vertex to $N_{\text{guard}}(v, w) \cap V(R)$. Hence, we get $|(N_{\text{exit}}(v, w) \cup N_{\text{guard}}(v, w)) \cap V(R)| \leq 2i$

Secondly, we determine the number of vertices in $N_{\text{prison}}(v, w) \cap V(R)$. Since G is reduced, by Remark 2.2.10, we know that these vertices need to be dominated by a single vertex in $N_{\text{guard}}(v, w) \cup N_{\text{prison}}(v, w)$. Moreover, since the region is simple, all vertices in $N_{\text{prison}}(v, w) \cap V(R)$ are neighbors of both v and w. By planarity, it follows that there can be at most 3 vertices in $N_{\text{prison}}(v, w) \cap V(R)$.

In summary, together with the vertices $v, w \in V(R)$, we get $|V(R)| \leq 5 + 2i$. □

Chapter 3

Bounded Search Trees

Systematic exhaustive search by a bounded search tree is probably the most commonly used technique to design "efficient" fixed-parameter algorithms. According to [157],

> *"[...] at least the majority of efficient FPT algorithms known so far are based on [...] bounded search trees [...]."*

The reason for this is quite obvious. Bounded search trees, in general, have proven to be particularly easy to describe and to implement, and—compared to other methods—no deep theoretical background knowledge is necessary to design a search tree algorithm. The analysis of the search tree size, however, might be complicated.

In Section 3.1, we give a brief introduction to bounded search trees and formalize their concept in a precise mathematical way (see Subsection 3.1.1). As two easy examples we exhibit the so-called "degree-branching" method for par-VERTEX COVER on general graphs and for par-INDEPENDENT SET on, e.g., graphs of bounded genus (see Subsection 3.1.2).

The main contribution of this chapter is a time $O(8^k n)$ search tree algorithm for par-DOMINATING SET on planar graphs (see Section 3.2). The search tree is particularly easy to describe, yet, the running time analysis is complicated. We observe that a direct application of the degree-branching method to par-DOMINATING SET will fail (see Subsection 3.2.1), forcing us to consider a so-called "annotated version" of par-DOMINATING SET. In order to guarantee a bounded branching degree in the search tree, some extra work is necessary. For that purpose, we introduce some reduction rules according to which the graph will be reduced in each search tree node (see Subsection 3.2.2). The technically most intricate part is to prove a new branching theorem which guarantees the existence of a low degree branching vertex in a reduced instance (see Subsection 3.2.3). Finally, it is observed that the branching theorem, in a sense, is optimal (see Subsection 3.2.4).

3.1 Background

Bounded search trees basically solve the problem on a given input instance in an exhaustive manner. More precisely, the central idea is to replace the current instance (I, k) by a set of instances $(I_1, k_1), \dots, (I_m, k_m)$ with smaller parameters, i.e., $k_i < k$ for all $1 \leq i \leq m$, and then to recursively apply this procedure to the replaced instances.

3.1.1 The Concept of Bounded Search Trees

The most important ingredient which already specifies a bounded search tree algorithm is given by what we call "branching rule." In the following definition, we use $\wp$ to denote the power set.

Definition 3.1.1 *Let $\mathcal{L} \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem. A mapping*

$$\Phi : \Sigma^* \times \mathbb{N} \to \wp(\Sigma^* \times \mathbb{N}), \quad (x, k) \mapsto \Phi((x, k)),$$

is called a branching rule *if*

(i) Φ *is computable in time polynomial in* $|x|$ *and* k.

(ii) $(x, k) \in \mathcal{L}$ *if and only if there exists some* $(x', k') \in \Phi((x, k))$ *such that* $(x', k') \in \mathcal{L}$.

(iii) *for all* $(x', k') \in \Phi((x, k))$, *we have* $k' < k$.

For a branching rule Φ and an instance (x, k), we define the branching vector[1] *to be the multiset*

$$\Delta_\Phi((x, k)) := \{\, k - k' \mid (x', k') \in \Phi((x, k)) \,\}.$$

A search tree algorithm for a parameterized problem $\mathcal{L}$ is specified by a branching rule Φ and some *"termination condition"* $\tau : \Sigma^* \to \{\text{TRUE}, \text{FALSE}\}$.[2] Using these specifications, the algorithm formally proceeds as shown in Fig. 3.1.

We now briefly discuss the algorithm scheme of Fig. 3.1. Since, in each step, the current problem parameter is reduced, the recursion in this algorithm finally terminates. For the correctness of the approach, it is clear that by property (ii) of Definition 3.1.1, we have $(x, k) \in \mathcal{L}$ if and only if $(x_i, k_i) \in \mathcal{L}$ for some instance (x_i, k_i) which is associated with a leaf node i of the search tree T as constructed in the algorithm in Fig. 3.1.

The size of the search tree clearly depends solely on the amount by which the problem parameter is reduced in each "branch," i.e., it depends on the branching vectors $\Delta_\Phi((x, k))$ that appear in T.

Suppose $\Delta := \{d_1, \dots, d_q\}$ is some fixed branching vector. Assuming that all branchings in the tree had this branching vector, an upper bound on the size of the search tree is easy to obtain by solving the homogeneous difference equation for the number of leaves in the tree:

$$S_k = \sum_{d \in \Delta} S_{k-d}, \quad \text{with } S_i = 1 \text{ for } i \leq 0.$$

[1] In the literature, the branching vector is the multiset $\Delta_\Phi((x, k))$ together with some ordering. However, since the ordering is irrelevant we prefer the notion as a multiset.

[2] A termination condition will always be easy compute. More precisely, we assume that on input (x, k) computing $\tau(x)$ and, if $\tau(x) = \text{TRUE}$, verifying whether $(x, k) \in \mathcal{L}$ can be done in time polynomial in $|x|$ and k.

```
procedure construct_search_tree (branching rule Φ, termination rule τ)
/* input:   an instance (x, k)                                   */
/* output:  a search tree for (x, k) specified by                */
/*          the branching rule Φ and the termination rule τ.     */
```

- create a tree $T = (V, F)$, where each node $i \in V$ of the tree is associated with a problem instance, *recursively* as follows:
  - associate the instance (x, k) with the root of T.
  - let i be a node with which we associate the instance (x_i, k_i)

    Case 1: `if` $k_i > 0$ `and` $\tau(x_i) = \text{FALSE}$ `then`
    - apply the branching rule Φ to (x_i, k_i). Let $j_i = |\Phi((x_i, k_i))|$.
    - create children $i_1, \ldots, i_{j_i}$ of i.
    - associate the j_i instances of $\Phi((x_i, k_i))$ with these nodes.

    Case 2: `if` $k_i \leq 0$ `or` $\tau(x_i) = \text{TRUE}$ `then` i is a leaf of the tree.

Figure 3.1: General scheme of a search tree algorithm specified by a branching rule Φ and termination condition τ.

Here S_ℓ denotes the number of leaves of the search tree under the assumption that the instance associated with its root has parameter value ℓ. The characteristic polynomial of this difference equation is

$$z^k - \sum_{d \in \Delta} z^{k-d}. \tag{3.1}$$

According to the general theory of difference equations (see [103, Chapter 2]), if c is a root of the polynomial in (3.1) with maximum absolute value among all other roots, then S_k is of the form $O(c^k)$ up to a polynomial factor and c is called the *branching number* that corresponds to the branching vector Δ. We denote this branching number by $\text{bn}(\Delta)$. Moreover, if c is a single root, then even $S_k = O(c^k)$.[3] Since, the size of the search tree is determined by the branching vector with the biggest branching number, we let

$$\text{bn}(\Phi, \tau) := \max\{\,\text{bn}(\Delta_\Phi((x, k))) \mid (x, k) \in \Sigma^* \times \mathbb{N},\ k > 0,\ \tau(x) = \text{FALSE}\,\} \tag{3.2}$$

denote the *global branching number*. The above observations are summarized in the following lemma.

Lemma 3.1.2 *Let $\mathcal{L}$ be a parameterized problem for which a search tree algorithms is specified by a branching rule Φ and a termination condition τ. Then the running time on input instance (x, k) is given by*

$$O(\text{bn}(\Phi, \tau)^k \cdot p(|x|, k)),$$

where $p(|x|, k)$ is the time needed to perform Φ on (x, k), to compute $\tau(x)$, and to verify whether $(x, k) \in \mathcal{L}$ in case $\tau(x) = \text{TRUE}$.

[3] For the branching vectors that appear in our setting, c is always real and will always be a single root.

3.1.2 The Degree-Branching Method

In this subsection, as a "warm-up," we provide some easy examples of search tree algorithms for par-VERTEX COVER, on the one hand, and for par-INDEPENDENT SET, on the other hand. Both examples have in common that the branching rule which specifies the search tree algorithm is based on the investigation of the closed neighborhood $N[v]$ of some vertex v. Since the branching number will depend on the degree of this vertex v, we call such a strategy "degree-branching."

3.1.2.1 An Easy Search Tree for par-VERTEX COVER

The base for a simple search tree algorithm lies in the following easy observation.

Observation 3.1.3 *Let V' be a minimum vertex cover of a graph $G = (V, E)$. Then, for every vertex $v \in V$, either $v \in V'$ or $N(v) \subseteq V'$. Moreover, for every vertex $v \in V'$, we have* $\mathrm{vc}(G) = \mathrm{vc}(G - v) + 1$

This suggests to choose a vertex v (of degree at least one) in each branch, and put either v or all of its neighbors $N(v)$ in the vertex cover we seek for. The parameter is reduced by one in the first case and by $|N(v)|$ in the latter case. We will stop if no further edge needs to be covered, i.e., we use the termination condition $\tau(G) = \text{TRUE}$ if and only if $E(G) = \emptyset$. In the following we only deal with graph instances G for which $\tau(G) = \text{FALSE}$. The branching rule formally reads as follows

$$\Phi((G, k)) := \left\{(G - v, k - 1), (G - N(v), k - \deg_G(v))\right\}, \tag{3.3}$$

where v is some vertex in $V(G)$ with $\deg_G(v) > 0$. The corresponding branching vector clearly is $\Delta_d := \{1, d\}$ where $d = \deg_G(v)$. The branching number $\mathrm{bn}(\Delta_d)$ satisfies the equation

$$z^{d-1}(z - 1) \ = \ 1, \tag{3.4}$$

and, hence, we get $\mathrm{bn}(\Delta_d) > \mathrm{bn}(\Delta_{d+1})$ for $d \geq 1$.

This means that the higher the degree $\deg_G(v)$ of the chosen vertex v, the better the branching number will be. As a consequence, the global branching number (see Eq. (3.2)) is given by $\mathrm{bn}(\Phi, \tau) = \mathrm{bn}(\Delta_1) = 2$ which means that this search tree algorithm has running time $O(2^k n)$ according to Lemma 3.1.2.

To further improve the running time, we observe that a vertex v with $\deg_G(v) = 1$ can be treated in a very simple way without any branching. Since it is always optimal to take $N(v)$ for such a vertex we improve the branching rule using the following case distinction.

Case 1: If there exists a vertex $v \in V(G)$ with $\deg_G(v) = 1$ then $\Phi((G, k)) := \{(G - N(v), k - 1)\}$.

Case 2: If all vertices in $V(G)$ have degree more than one then
$\Phi((G, k)) := \{(G - v, k - 1), (G - N(v), k - \deg_G(v))\}$, for some $v \in V(G)$.

Case 1 has branching vector $\{1\}$ with corresponding branching number 1. The worst-case branching number in Case 2 is achieved if we use a vertex v with $\deg_G(v) = 2$, we now get a branching number $\mathrm{bn}(\Delta_2) \approx 1.618$ (which is a root of Eq. (3.4) with $d = 2$), implying a time $O(1.618^k|G|)$ algorithm.

We further improve the running time if we find a suitable branching for a vertex of degree two, which has a smaller branching number than $\mathrm{bn}(\Delta_3) \approx 1.4656$. Suppose v is a vertex with neighbors $N(v) = \{v_1, v_2\}$. Again we branch by either choosing $N(v)$ to be in the vertex cover or v itself. Note that it is always at least as good to take $N(v)$ in the vertex cover as v together with any one of the vertices from $N(v)$. Hence, if we use v in one branch, we may delete $N(v)$ and use $N(N(v))$ in order to cover all edges between $N(v)$ and $N(N(v))$. If we assume that there exists no vertex of degree one then we know that $|N(N(v))| \geq 2$. This results in the following more advanced branching rule.

Case 1: If there exists a vertex $v \in V(G)$ with $\deg_G(v) = 1$ then $\Phi((G,k)) := \{(G-N(v), k-1)\}$.

Case 2: If there exists a vertex $v \in V(G)$ with $\deg_G(v) = 2$ then
$\Phi((G,k)) := \{(G - N(N(v)), k - |N(N(v))|), (G - N(v), k - 2)\}$.

Case 3: If all vertices in $V(G)$ have degree at least three then
$\Phi((G,k)) := \{(G - \{v\}, k - 1), (G - N(v), k - \deg_G(v))\}$, for some $v \in V(G)$.

The worst-case branching vector for Case 2 is given by $\{2, 2\}$ (if $|N(N(v))| = 2$), and the worst-case branching vector for Case 3 now is $\Delta_3 = \{1, 3\}$. Since, $\mathrm{bn}(\{2, 2\}) = \sqrt{2}$ (which is a root of the polynomial $z^2 - 2 = 0$) and $\mathrm{bn}(\Delta_3) \approx 1.4656$ (which is a root of the polynomial in Eq. (3.4) with $d = 3$), we obtain an algorithm with running time $O(1.4656^k|G|)$.

This "game" of steadily refining the branching rule can be continued, at each step making the case distinction more and more intricate. In the literature, we find a whole sequence of contributions (see [32, 156, 180, 56]) in this direction with a current record base of approximately 1.2852.

3.1.2.2 An Easy Search Tree for par-Independent Set

The statement corresponding to Observation 3.1.3, in the case of par-independent set, reads as follows.

Observation 3.1.4 *Let V' be a maximum independent set of a graph $G = (V, E)$. Then, for every vertex $v \in V$, at least one vertex from $N[v]$ belongs to V'. Moreover, for every vertex $v \in V'$, we have* $\mathrm{is}(G) = \mathrm{is}(G - N[v]) + 1$.

With this observation at hand, it is clear that, for a given vertex v of a graph G, we decide to choose, in each branch, a vertex u from $N[v]$ to be in the independent set we seek for. If we decide to take u in the independent set, we reduce the problem parameter by one, and we have to remove u together with all neighbors of u from the graph. Formally, the branching rule is given by

$$\Phi((G,k)) := \bigcup_{u \in N[v]} \big\{(G - N[u], k - 1)\big\} \tag{3.5}$$

for some vertex $v \in V(G)$. We call v the *branching vertex*. Suppose the degree of the branching vertex v is $d := \deg_G(v)$, then the corresponding branching vector is the multiset $\Delta_d := \{1, \dots, 1\}$ with $|N[v]| = d + 1$ elements. Consequently, the branching number is $\mathrm{bn}(\Delta_d) = d + 1$. As a termination condition we use $\tau(G) = \text{TRUE}$ if and only if $V(G) = \emptyset$.

Unfortunately, if we try to construct a search tree specified by the branching rule Φ and the termination condition τ, the global branching number $\mathrm{bn}(\Phi, \tau)$ (see Eq. (3.2)) may be unbounded since there is no constant c such that an arbitrary graph G always contains a branching vertex of degree bounded by c.

On a suitably restricted graph class, however, the above given degree-branching yields a bounded search tree.

Lemma 3.1.5 *Suppose we are given a graph class $\mathbb{G}$ that is closed under taking subgraphs and that guarantees a vertex of degree d for some constant d.*[4] *Then* par-INDEPENDENT SET *on $\mathbb{G}$ can be solved in time*

$$O(\,(d+1)^k |G|\,),$$

using a bounded search tree algorithm. Here, k is the size of the independent set we seek for.

Proof: We use a search tree specified by the termination rule τ above and the branching rule given in (3.5). Here, for a graph G, the branching vertex v to be chosen is such that $\deg_G(v) \leq d$. By assumption such a vertex always exists in G. By the discussion preceding this lemma we obtain $\mathrm{bn}(\Phi, \tau) = d + 1$. Lemma 3.1.2 implies the claimed running time, since Φ can be carried out in time $O(|G|)$. □

We finish the discussion by providing an example of a graph class which fulfills the assumptions of Lemma 3.1.5. Recall from Section 1.2 the class $\mathbb{G}(\mathcal{S}_g)$ of graphs that admit a crossing-free embedding on the surface S_g of genus g.

Lemma 3.1.6 *A graph $G \in \mathbb{G}(\mathcal{S}_g)$ always contains a vertex of degree bounded by*

$$d_g := \lceil 2(1 + \sqrt{3g+1}) \rceil \quad \textit{for} \ \ g > 0 \quad \textit{and} \ \ d_0 := 5.$$

Proof: The proof is deferred to the Appendix at the end of this chapter. □

Remark 3.1.7 We remark that the value $d_g = \lceil 2(1 + \sqrt{3g+1}) \rceil$ is optimal in the following sense: Let $d_g^{\mathrm{opt}} := \max_{G \in \mathbb{G}(\mathcal{S}_g)} \min_{v \in V(G)} \deg_G(v)$ be the smallest number such that each graph in $\mathbb{G}(\mathcal{S}_g)$ admits a vertex of degree at most d_g^{opt}. Then, it holds that there is a sequence $(g_n)_{n \in \mathbb{N}}$ such that $d_{g_n} = d_{g_n}^{\mathrm{opt}}$ for all $n \in \mathbb{N}$.

To see this, first observe that by Lemma 3.1.6, we have $d_g^{\mathrm{opt}} \leq d_g$. Conversely, by a famous result of Ringel and Youngs [168], we know that the genus of the complete graph K_n with n vertices is

$$g(K_n) = \lceil (n-3)(n-4)/12 \rceil\,.$$

On the one hand, some easy arithmetic shows that $d_{g(K_n)} \leq n-1$ if $n \geq 15$. On the other hand, since all vertices in K_n have degree $n-1$, we obtain $n-1 \leq d_{g(K_n)}^{\mathrm{opt}}$. This gives us $d_g^{\mathrm{opt}} = d_g$ for all $g \in \{g(K_n) \mid n \geq 15\}$.

[4]This means that, for each $G \in \mathbb{G}$, there exists a $v \in V(G)$ with $\deg_G(v) \leq d$.

3.2 A Bounded Search Tree for par-ANNOTATED DOMINATING SET on Planar Graphs

The search trees for par-VERTEX COVER and par-INDEPENDENT SET from the previous subsection were easy to derive. In the case of par-DOMINATING SET the situation is much more intricate. We will see in the following that we encounter difficulties when directly carrying over the degree-branching method to the par-DOMINATING SET problem. In search for an analogue to Observation 3.1.4, we find the following.

Observation 3.2.1 *Let* V' *be a minimum dominating set of a graph* $G = (V, E)$*. Then, for every vertex* $v \in V$*, at least one vertex from* $N[v]$ *belongs to* V'*.*

This again suggests to perform a degree-branching for some branching vertex v, i.e., to choose—in each branch—some vertex $u \in N[v]$ to belong to the dominating set. However, it is not clear, in which way the graph must be modified in a branch corresponding to the choice of u:

On the one hand, if we decide to choose u to belong to the dominating set, we might be tempted to remove $N[u]$ from the graph, since these vertices are already dominated. This would correspond to what was done by the branching rule Φ as defined in the assignment of (3.5) for par-INDEPENDENT SET. However, this mapping fails to be a branching rule in the case of par-DOMINATING SET. This is due to the fact that with the choice of u being in the dominating set the neighbors $N(u)$ still are suitable candidates for an optimal dominating set, and, hence, *must not* be removed from the graph.[5]

On the other hand, if we decide to choose u to belong to the dominating set and still want to preserve the possibility of taking one of its neighbors, we might consider the mapping

$$\Phi((G, k)) := \bigcup_{u \in N[v]} \{(G - u, k - 1)\}.$$

Here, in each branch, instead of removing the closed neighborhood $N[u]$, we only remove u from the graph. Again, this is *not* a branching rule for par-DOMINATING SET, since—in the graph $G - u$—we lose the information that with the choice of u being in the dominating set, the vertices in $N(u)$ are already dominated.[6]

These considerations lead us to formulate a generalization of the DOMINATING SET problem, where there are two kinds of vertices in our graph: “regular” vertices which still need to be dominated and vertices which are already dominated but still are suitable candidates to be chosen for the dominating set we seek for.

[5] One may verify, that, unlike in the situation for par-INDEPENDENT SET (see Observation 3.1.4), for a vertex u that belongs to an optimal dominating set, $\mathrm{ds}(G) < \mathrm{ds}(G - N[u]) + 1$ may well happen.

[6] One may verify, that $\mathrm{ds}(G) < \mathrm{ds}(G - u) + 1$ is possible for a vertex u that belongs to an optimal dominating set.

3.2.1 Degree-Branching for ANNOTATED DOMINATING SET

We use the following generalization of DOMINATING SET.

Definition 3.2.2 ANNOTATED DOMINATING SET *is the optimization problem* $(I, F, c, \min)$ *which is defined as follows:*

(i) *The set of instances* I *consists of all graphs* $G = (B\dot{\cup}W, E)$ *with two disjoint sets* B *of black and* W *of white vertices. Such graphs will be called* black and white graphs*.*

(ii) *A feasible solution for a black and white graph is a set* V' *of* $B \cup W$ *which dominates all black vertices, i.e.,* $B \subseteq N[V']$*. We call such a set* V' *an* annotated dominating set*.*

(iii) *The cost for a feasible solution* V' *is given by* $c_Q(V') := |V'|$*.*

We denote by $\text{ads}(G)$ *the size of the smallest annotated dominating set in a black and white graph* G*.*

In other words, only black vertices need to be dominated in this setting. In this sense, a white vertex can be interpreted as a vertex which is already dominated but still is a valid choice to be in the (annotated) dominating set.

In the rest of this section we will establish the following main result. We follow parts of [6].

Theorem 3.2.3 par-ANNOTATED DOMINATING SET *on planar graphs can be solved, using a bounded search tree algorithm, in time* $O(8^k n)$*, where* n *is the number of vertices of an input graph and* k *is the size of the (annotated) dominating set we seek for.*

In combination with the linear problem kernel (see Theorem 2.2.1) we get the following.

Corollary 3.2.4 par-DOMINATING SET *on planar graphs can be solved in time* $O(8^k k + n^3)$*.*

Remark 3.2.5 Downey and Fellows [80, 82] claimed a search tree algorithm for par-ANNOTATED DOMINATING SET running in time $O(11^k n)$. The analysis of their algorithm, however, turned out to be flawed. Hence, Theorem 3.2.3 seems to give the first correctly analyzed fixed-parameter algorithm for par-DOMINATING SET on planar graphs with running time $O(c^k n)$ for *small* constant c that even improves the previously claimed constant.

Discussion of the proof of Theorem 3.2.3. We will pursue a degree-branching strategy similar to the one presented for par-INDEPENDENT SET in Subsection 3.1.2.2. Indeed, for par-ANNOTATED DOMINATING SET, we have a complete analogy to Observation 3.1.4.

Observation 3.2.6 *Let* V' *be a minimum annotated dominating set of a black and white graph* $G = (B\dot{\cup}W, E)$*. Then, for every vertex* $v \in B$*, at least one vertex from* $N[v]$ *belongs to* V'*. Moreover, for every vertex* $v \in V'$*, we have* $\text{ads}(G) = \text{ads}(G \ominus v) + 1$*.*

Here, we define $G \ominus v$ *to be the black and white graph that is obtained by removing* v *(together with all adjacent edges) and letting all neighbors* $N(v)$ *be white.*

Proof: It is obvious that at least one vertex from $N[v]$ belongs to V'. (compare with Observation 3.2.1). We show that $\text{ads}(G) = \text{ads}(G \ominus v) + 1$ if v belongs to V'. Suppose V'' is a minimum annotated dominating set of $G \ominus v$. Then, clearly, $V'' \cup \{v\}$ is an annotated dominating set for G and, hence, $\text{ads}(G) \leq \text{ads}(G \ominus v) + 1$. Conversely, in order to show that $\text{ads}(G) \geq \text{ads}(G \ominus v) + 1$, we claim that $V' \setminus \{v\}$ is an annotated dominating set for $G \ominus v$. To see this, observe that a black vertex u in $G \ominus v$ is at distance at least two to v in G (since $N(v)$ are all white vertices in $G \ominus v$ by definition of the operation "$\ominus$"). This implies that u must be dominated by some vertex in $V' \setminus \{v\}$ in G and, hence, in $G \ominus v$. □

As a result, the mapping

$$\Phi((G, k)) := \bigcup_{u \in N[v]} \{(G \ominus u, k-1)\}, \tag{3.6}$$

where v is some *black* vertex v in the black and white graph G, is a branching rule for par-ANNOTATED DOMINATING SET. If $\deg_G(v) = d$, the branching vector is the multiset $\Delta_d = \{1, \ldots, 1\}$ of order $d + 1$ with the branching number $\text{bn}(\Delta_d) = d + 1$ (as defined in Subsection 3.1.1). Since we require to dominate all black vertices only, we use the termination rule $\tau(G = (B \dot\cup W, E)) = \text{TRUE}$ if and only if $B = \emptyset$.

As in Subsection 3.1.2.2, the global branching number $\text{bn}(\Phi, \tau)$ may be unbounded since there is no constant c such that an arbitrary black and white graph G always contains a black branching vertex of degree bounded by c.

Clearly, if we restrict ourselves to the class $\mathbb{G}_{\deg \leq d}$ of (black and white) graphs of bounded degree d, then $\text{bn}(\Phi, \tau) \leq d+1$ and we obtain a time $O((d+1)^k n)$ algorithm for par-ANNOTATED DOMINATING SET on $\mathbb{G}_{\deg \leq d}$.

An extension of this result to planar graphs, however, is **not** immediately possible. The reason for this is that—even though every planar graph has a vertex of degree at most 5—for a black and white graph, it is not necessarily the case that we find a *black* vertex of degree bounded by 5 that could be used as a branching vertex. A simple counterexample is, e.g., given by a star which n vertices consisting of a single black vertex of degree $n-1$ and $n-1$ white vertices of degree one. In this case, branching rule (3.6) would create a branch with branching number n (sic!).

This shows that establishing a time $O(c^k n)$ search tree algorithm for par-ANNOTATED DOMINATING SET on planar graphs needs some extra work that avoids such situations. To this end, in each node of the search tree, we will transform the current instance into an instance that is equivalent in terms of being member of the language, but which guarantees a *black* branching vertex of bounded degree. More precisely, in Subsection 3.2.2 and Subsection 3.2.3 we will prove that there exists a transformation Γ on the set of black and white graphs which can be computed in linear time, such that, for a black and white graph G,

(i) $\text{ads}(G) = \text{ads}(\Gamma(G))$ (see Lemma 3.2.9), and

(ii) there exists a *black* vertex of degree at most 7 in $\Gamma(G)$ (see Corollary 3.2.11).

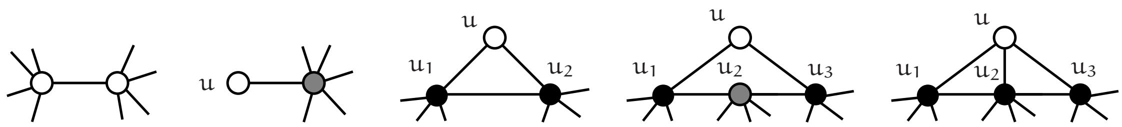

Figure 3.2: Illustration of the settings in which bw-Rules 1, 2, 3, and 4 apply.

Combining Γ and the mapping Φ from (3.6) we obtain the branching rule

$$\Psi : \quad (G, k) \mapsto \bigcup_{u \in N_{\Gamma(G)}[v]} \{(\Gamma(G) \ominus u, k-1)\}, \tag{3.7}$$

where v is chosen to be a black vertex in $V(\Gamma(G))$ with $\deg_{\Gamma(G)}(v) \le 7$.

From the previous discussion it is clear the search tree specified by Ψ and the termination rule τ has global branching number $\mathrm{bn}(\Psi, \tau) \le 8$ (see Eq. (3.2)). Since Ψ can be computed in linear time, Theorem 3.2.3 follows from Lemma 3.1.2. □

3.2.2 Further Reduction Rules

In order to define the transformation Γ mentioned in the previous section, we consider the following reduction rules for simplifying a black and white graph instance of the ANNOTATED DOMINATING SET problem (see Fig. 3.2 for an illustration).

bw-Rule 1 *Delete edges between white vertices.*

bw-Rule 2 *Let u be a white vertex u of degree 1. Then, delete u.*

bw-Rule 3 *Let u be a white vertex u of degree 2, with two black neighbors u_1 and u_2.*

(3.1) *If u_1 and u_2 are connected by an edge, then delete u.*

(3.2) *If u_1 and u_2 are connected via a third (black or white) vertex u_3, then delete u.*

bw-Rule 4 *Let u be a white vertex of degree 3, with three black neighbors u_1, u_2 and u_3. If the edges $\{u_1, u_2\}$ and $\{u_2, u_3\}$ are present in G (and possibly also $\{u_1, u_3\}$), then delete u.*

The following result shows that all reduction rules are sound.

Lemma 3.2.7 *Let G be a black and white graph and let G' be the resulting graph after having applied bw-Rule i (for some $1 \le i \le 4$) to G. Then $\mathrm{ads}(G) = \mathrm{ads}(G')$.*

Proof: bw-Rule 1 is correct since a vertex set D is an annotated dominating set for G if and only if it is an annotated dominating set for G'.

For all other rules, observe that removing a white vertex u always implies that $\mathrm{ads}(G) \leq \mathrm{ads}(G-u)$, hence, for the remaining rules, we only need to show that $\mathrm{ads}(G) \geq \mathrm{ads}(G-u)$ holds. We assume now that D is an annotated dominating set for G. We show that, in all remaining cases, we can construct an annotated dominating set of the same size. Let u be the white vertex considered in the corresponding rule. If $u \notin D$ then D also is an annotated dominating set for G'. Suppose that $u \in D$.

In the case of bw-Rule 2 let $D' := (D \setminus \{u\}) \cup N_G(u)$.

In the case of bw-Rule 3.1 let $D' := (D \setminus \{u\}) \cup \{u_1\}$.

In the case of bw-Rule 3.2 let $D' := (D \setminus \{u\}) \cup \{u_3\}$.

In the case of bw-Rule 4 let $D' := (D \setminus \{u\}) \cup \{u_3\}$.

It is easy to verify that in all of the cases the set D' is an annotated dominating set for G' with $|D'| = |D|$. □

Definition 3.2.8 *Let* G *be a black and white graph such that bw-Rule* i *cannot be applied (*$1 \leq i \leq 4$*). Then we say that* G *is* reduced *with respect to this rule.*

Given a black and white graph G, we can apply bw-Rule 1 to all edges between white vertices and then apply bw-Rules 2, 3, and 4 (if possible) to all white vertices of degree one, two, and three to obtain a black and white graph G' which is reduced with respect to all these rules. Below, we will see that reducing a graph with respect to bw-Rules 1, 2, 3.1, and 4 can be done in linear time. It is not clear, however, how to reduce a graph with respect to bw-Rule 3.2 in linear time.[7] Here, a certain technicality comes into play to keep the overall running time linear (see [105]): When transforming G into G', we only apply bw-Rule 3.2 to white vertices u with neighbors $N(u) = \{u_1, u_3\}$, for which either u_1 or u_3 has at most seven neighbors that have degree at least 4.

Let Γ be the transformation which maps G to G'.

Lemma 3.2.9 *The transformation* Γ *can be computed in linear time. And, for a black and white graph* G*, we have* $\mathrm{ads}(G) = \mathrm{ads}(\Gamma(G))$*.*

Proof: The fact that $\mathrm{ads}(G) = \mathrm{ads}(\Gamma(G))$ follows from Lemma 3.2.7.

We now argue that reducing a graph with respect to bw-Rules 1, 2, 3.1, 3.2 (when applied only to white vertices u, for which either u_1 or u_3 has at most seven neighbors that are of degree at least 4) and 4 can be done in linear time.

In case of bw-Rule 1 and bw-Rule 2 this is clear, since all edges between white vertices and all white vertices of degree one can be removed by a single scan of the graph.

In the case of bw-Rule 3.1, for each white vertex of degree two, we determine the neighbors u_1 and u_2 and ask the query whether $\{u_1, u_2\}$ is an edge in G. If this is the case, we remove u. In total we have to answer at most $O(n)$ queries of this form, which can be done in linear time by sorting the edges and the queries via radix sort.

[7] It was stated in [6] that a graph can be reduced with respect to all rules in linear time. Thanks go to Torben Hagerup who noticed this inaccuracy and suggested a fix of the flaw [105].

In the case of bw-Rule 4, for each white vertex of degree three, we determine the neighbors u_1, u_2 and u_3 and ask the three queries whether any of the sets $\{u_i, u_j\}$ ($1 \leq i, j \leq 3$, $i \neq j$) is an edge in G. If two of these queries are answered positively, we remove u. In total we have at most $O(n)$ queries of this form, which can be answered in linear time.

In the case of bw-Rule 3.2, for each white vertex of degree two, we determine the neighbors u_1 and u_3 of u and check whether one of these vertices has at most seven neighbors that are of degree at least 4. (Observe that, for fixed u, this can be done in constant time since we only need to determine the degree of u_1 and u_3 in the graph $G - \{v \in V(G) : \deg_G(v) \leq 4\}$. These degrees could have been determined in a preprocessing step in linear time.) If this is not the case (i.e., if both vertices u_1 and u_3 have more than seven neighbors of degree at least 4), we leave the graph unchanged. Otherwise (assuming, w.l.o.g, that u_1 has at most seven such neighbors) we have to check whether u_1 is connected to u_3 by a vertex v. To answer this, we ask for each of the at most seven such neighbors v, the queries whether $\{v, u_2\}$ is an edge in G. If one such query is answered positively, we remove u. In total we have at most $O(n)$ queries of this form, which can be answered in linear time. It remains to check, whether u_1 is connected to u_3 by a vertex v of degree two or three. To cover these cases, we check for each vertex v of degree two or three, whether there are two neighbors u_1 and u_3 of v (among which one vertex has to have at most seven neighbors of degree at least four and) which are connected by a white vertex of degree two. This needs at most three queries per vertex v, meaning that the total number of queries again is linear and, hence, can be answered in linear time. □

3.2.3 A New Branching Theorem

In the course of this subsection, we will prove the following main theorem.

Theorem 3.2.10 *If* $G = (B \dot{\cup} W, E)$ *is a connected planar black and white graph that is reduced with respect to bw-Rules 1, 2, 3, and 4, then there exists a black vertex* $v \in B$ *with* $\deg_G(v) \leq 7$.

In particular, this theorem implies that our transformation Γ (defined in the previous subsection) fulfills the desired property.

Corollary 3.2.11 *Let* G *be a black and white graph, then* $\Gamma(G)$ *contains a black vertex of degree at most 7.*

Proof: Let G' be the graph obtained when reducing G with respect to bw-Rules 1, 2, 3, and 4. In particular, each connected component of G' is reduced. Hence, there exists a black vertex v with $\deg_{G'}(v) \leq 7$ (in one such component). The only difference between G' and $\Gamma(G)$ is that $\Gamma(G)$ may contain white vertices of degree two where both neighbors have more than seven neighbors that are of degree at least 4. We argue that $\deg_{\Gamma(G)}(v) \leq 7$. If this were not the case, then v must have additional neighbors which are not present in G'. By the above observation an additional neighbor must be a white vertex u of degree two where both neighbors (in particular, the neighbor v) have more than seven neighbors that are of degree at least 4. Hence, there exist vertices $v_1, \ldots, v_\ell \in N_{\Gamma(G)}(v)$ ($\ell \geq 8$) which are of degree at least 4. Since these vertices are not removed by any of the reduction rules, it follows that $v_1, \ldots, v_\ell \in N_{G'}(v)$ which implies $\deg_{G'}(v) > 7$, a contradiction. □

In the remainder of this subsection we give a proof of our Branching Theorem 3.2.10. To this end, firstly, using Euler's formula for planar graphs, we establish a criterion which guarantees a black vertex of degree at most 7 (see Subsection 3.2.3.1). Secondly, in order to verify this criterion (see Subsection 3.2.3.3) for a reduced black and white graph $G = (B\dot{\cup}W, E)$, we have to analyze the black induced subgraph $G[B]$ (see Subsection 3.2.3.2). This latter part is technically demanding.

3.2.3.1 A Technical Lemma

The following technical lemma is based on Euler's formula (see Eq. (1.2)).

Lemma 3.2.12 *Suppose* $G = (B\dot{\cup}W, E)$ *is a connected plane black and white graph with* b *black vertices,* w *white vertices, and* m *edges. Let the subgraph induced by the black vertices be denoted* $H = G[B]$*. Let* c_H *denote the number of components of* H *and let* f_H *denote the number of faces of* H*. Let*

$$z = \big(3(b+w) - 6\big) - m \tag{3.8}$$

measure the extent to which G *fails to be a triangulation of the plane. If the criterion*

$$3w - 4b - z + f_H - c_H < 7 \tag{3.9}$$

is satisfied, then there exists a black vertex $v \in B$ *with* $\deg_G(v) \leq 7$*.*

Proof: Let the (total) numbers of vertices, edges and faces of G be denoted n, m, f, respectively. Let m_{bw} be the number of edges in G between black and white vertices, and let m_{bb} denote the number of edges between black and black vertices. With this notation, we have the following relationships.

$$\begin{aligned}
n - m + f &= 2 && \text{(Euler's formula for } G \text{ (connected))} && (3.10)\\
n &= b + w && && (3.11)\\
m &= m_{bb} + m_{bw} && && (3.12)\\
b - m_{bb} + f_H &= 1 + c_H && \text{(Euler's formula for } H \text{ (need not be connected))} && (3.13)\\
2n - 4 - z &= f && \text{(by Eq. (3.8), (3.10), and (3.11))} && (3.14)
\end{aligned}$$

If the lemma were false, then we would have, using (3.12),

$$8b \;\leq\; 2m_{bb} + m_{bw} \;=\; m_{bb} + m. \tag{3.15}$$

We will assume this and derive a contradiction. The following inequality holds:

$$\begin{aligned}
3 + c_H &\leq n + b - (m_{bb} + m) + f + f_H && \text{(by (3.10) and (3.13))}\\
&\leq n + b - 8b + f + f_H && \text{(by (3.15))}\\
&= 3n - 7b + f_H - 4 - z && \text{(by (3.14))}\\
&= 3w - 4b + f_H - 4 - z. && \text{(by (3.11))}
\end{aligned}$$

This yields a contradiction to (3.9). □

In the end, we will use criterion (3.9) to establish Theorem 3.2.10.

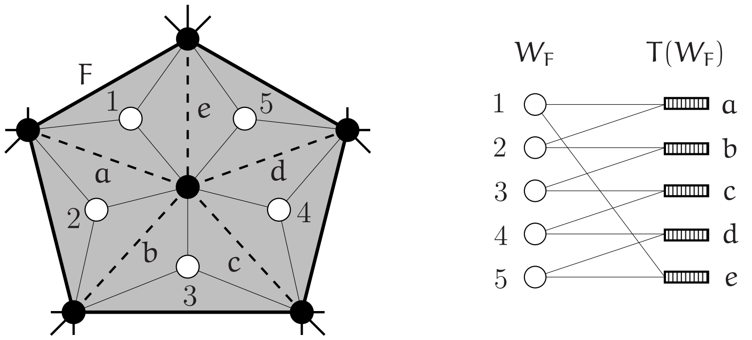

Figure 3.3: The left-hand diagram shows a (very simple) example of a face F of a reduced black and white graph with white vertices $\{1, 2, 3, 4, 5\}$. Note that $G[B]$ is not connected in this example. More precisely, we get $c_F = 1$. The dotted lines $\{a, b, c, d, e\}$ correspond to possible black-black edges that still can be drawn inside F in order to triangulate that part of $G[B]$. Hence, we have $w_F = t_F = 5$. Besides, one can verify that $z_F = 5$. The right-hand diagram shows the construction of the bipartite graph $H(W_F)$ that is needed in the proof of Proposition 3.2.15.

3.2.3.2 Analyzing the Black Subgraph

Let us consider some fixed crossing-free embedding of G in the plane.[8]

For each face F, of the subgraph $G[B]$ induced by the black vertices, we will determine the number of white vertices that can possibly sit inside F.

Notation: Let $G = (B \dot{\cup} W, E)$ be a plane black and white graph and let $\mathcal{F}$ be the set of faces of $G[B]$. Then, for each $F \in \mathcal{F}$, we let

- w_F denote the number of white vertices sitting inside F,
- z_F denote the number of edges that would have to be added in order to complete a triangulation of the subgraph of G inside F (including the boundary of F),
- t_F denote the number of edges needed to triangulate F in $G[B]$ (that is, triangulating only between the black vertices on the boundary of F, and noting that the boundary of F may not be connected), and
- c_F denote the number of components of the boundary of F, minus 1.

We refer to the left-hand graph in Fig. 3.3 for an example of the above introduced notion.

The following lemma will be useful for our further analysis.

[8] Again, for the ease of presentation, since it will always be clear from the context, in the following, we will not distinguish between a vertex $v \in V$ and the point $\phi(v)$ in the plane, or an edge $e \in E$ and the arc $\phi(e)$. Moreover, we simply write G for the plane graph (G, ϕ).

Lemma 3.2.13 *Let* $G = (V_1 \dot{\cup} V_2, E)$ *be a plane graph, where both* $G[V_1]$ *and* $G[V_2]$ *are forests. Then, the vertices of* V_1 *can be connected in a treelike fashion without destroying planarity. The number of added edges equals the number of components of* $G[V_1]$ *minus one.*

Proof: The proof is deferred to the Appendix at the end of this chapter. □

The following properties give an interrelation between the numbers w_F, t_F, and z_F.

Proposition 3.2.14 *Let* $G = (B \dot{\cup} W, E)$ *be a plane black and white graph that is reduced with respect to bw-Rule 1, and let* F *be a face of* $G[B]$*. Then, using the notation above, we have*

$$w_F + c_F \le z_F + 1.$$

Proof: Consider the "face-graph" $G_F := G[B_F \cup W_F]$, where B_F is the set of black vertices forming the boundary of F and W_F is the set of white vertices inside F. Note that G_F may consist of several "black components," connected among themselves through white vertices. Contracting each of these black components into one (black) vertex, we obtain the *bipartite* black and white graph G'_F. Note that both the black and also the white vertices form independent sets in G'_F by the above construction and since G is assumed to be reduced with respect to bw-Rule 1. Clearly, G'_F is still planar. Since G'_F is a bipartite planar graph, the assumptions of Lemma 3.2.13 are fulfilled (with V_1 being the white vertices and V_2 being the black vertices) and we can connect the white vertices among themselves by a tree of $w_F - 1$ white-white edges. Observe that the resulting black and white graph G' again satisfies the assumptions of Lemma 3.2.13 (now, V_1 are the black vertices and V_2 are the treelike connected white vertices). Thus, in addition, we can connect the black vertices among themselves by a tree of c_F black-black edges. Clearly, this implies that we can also add at least $c_F + w_F - 1$ new edges to G_F without destroying planarity. Hence, we need at least $c_F + w_F - 1$ additional edges to triangulate the interior of F in the graph G. □

The proof of the following proposition is technically involved. The main difficulty arises from the fact that the subgraph $G[B]$ need not be connected. To get an insight in the proof strategy, and for the ease of a more fluent presentation, we restrict ourselves to giving a proof for the special case when $G[B]$ *is* connected. For a full proof of the more general situation, the reader is deferred to the Appendix at the end of this chapter.

Proposition 3.2.15 *Suppose* $G = (B \dot{\cup} W, E)$ *is a plane black and white graph which is reduced with respect to bw-Rule 4. And suppose that* $\deg(u) \ge 3$ *for all* $u \in W$*. Let* F *be a face of* $G[B]$*. Then, using the notation above, we have*

$$w_F \le t_F.$$

Proof (Sketch): As already mentioned, we only give a proof under the assumption that $G[B]$ is connected. The general situation is treated in a similar way, however, more sophisticated technical details are needed (see the Appendix of this chapter).

Consider a face F of the black induced subgraph $G[B]$. Let $W_F \subseteq W$ be the set of white vertices in the interior of F, and let $B_F \subseteq B$ denote the black vertices on the boundary of F.

We want to find at least $w_F = |W_F|$ many black-black edges that can be added to $G[B]$ inside F without destroying planarity. For that purpose, define the set

$$E^{\text{poss}} := \big\{ e = \{b_1, b_2\} \mid b_1, b_2 \in B_F \ \wedge \ e \notin E(G[B]) \big\}$$

of non-existing black-black edges.

For a subset $W' \subseteq W_F$, we construct a bipartite graph

$$H(W') := (W' \dot\cup T(W'), E(W')) \tag{3.16}$$

as follows. In $H(W')$, the first bipartition set is formed by the vertices W' and the second one is given by the set

$$T(W') := \big\{ e = \{b_1, b_2\} \in E^{\text{poss}} \mid \exists u \in W' : e \subseteq N_G(u) \big\}.$$

The edges in $H(W')$ are then given by

$$E(W') := \big\{ \{u, e\} \mid u \in W', e \in T(W'), e \subseteq N_G(u) \big\}.$$

An example for the construction of $H(W')$ is illustrated in Fig. 3.3. In this way, the set $T(W')$ gives us vertices in $H(W')$ that correspond to pairs $e = \{b_1, b_2\}$ of black vertices in B_F between which we still can draw an edge in $G[B]$. Note that the edge e can even be drawn in the interior of F, since b_1 and b_2 are connected by a white vertex in $W' \subseteq W_F$. In particular, this shows that $|T(W_F)| \leq t_F$. Hence, if we can find a matching M in $H(W_F)$ which assigns to each vertex $u \in W_F$ an edge $M(u) \in T(W_F)$ we know that $w_F = |W_F| = |M(W_F)| \leq t_F$, which finishes the proof of this proposition. The existence of such a matching is due to the following claim (with $W' = W_F$).

Claim: For each set $W' \subseteq W_F$, there exists a W'-saturated matching in $H(W')$, i.e., a matching in $H(W')$ which uses all vertices in W'.

Proof of the Claim: We prove the claim by induction on the size of the subset $W' \subseteq W_F$. If $|W'| = 0$ there is nothing to prove. Now consider some set $W' \subseteq W_F$. We argue on the subgraph $G[B_F \cup W']$ of the "face graph" $G_F = G[B_F \cup W_F]$. Let $\langle b_1, \ldots, b_m \rangle$ be a clockwise representation of the face boundary B_F—with possible multiple enumeration of some vertices. As demonstrated in the proof of Proposition 3.2.14, we can connect the white vertices W' inside the face F by a tree like-structure. Let $u \in W'$ be a leaf of this tree and consider the neighbors $N(u)$ (in B_F). Since u is a leaf of the tree, there are two distinct vertices $b_{i_1}, b_{i_2} \in N(u) \cap B_F$, such that $N(u) \subseteq \{b_{i_1}, b_{i_1+1}, \ldots, b_{i_2}\}$ and no further vertex from W' lies in the region bounded by $\langle u, b_{i_1}, b_{i_1+1}, \ldots, b_{i_2}, u \rangle$. In particular, due to planarity, no further vertex from W' (except for u) has a neighbor in $\{b_{i_1+1}, \ldots, b_{i_2-1}\}$. By assumption, u has at least three neighbors. Since G is reduced with respect to bw-Rule 4, there must exist a further neighbor $b_{i_3} \in \{b_{i_1+1}, \ldots, b_{i_2-1}\}$, such that either the edge $\{b_{i_1}, b_{i_3}\}$, or $\{b_{i_2}, b_{i_3}\}$ is not present in G (otherwise we could have applied bw-Rule 4 to u). W.l.o.g., assume that $e_u := \{b_{i_1}, b_{i_3}\} \in T(W')$.

By induction hypothesis, there exists a W'_0-saturated matching M_0 in the subgraph $H(W'_0)$ of $H(W')$. Note that no further vertex from $W'_0 := W' \setminus \{u\}$ shares both b_{i_1} and b_{i_3} as a neighbor, which means that $e_u \notin T(W'_0)$. Hence, $M \cup \{\{u, e_u\}\}$ is a W'-saturated matching of $H(W')$. $\square$

We remark that the upper bounds in Propositions 3.2.14 and 3.2.15 are sharp as can be seen from the graph in Fig. 3.3.

3.2.3.3 Proving the Branching Theorem

We now are ready to give a complete proof of Theorem 3.2.10. Proving this theorem by contradiction, it will be helpful to know that we can assume that a corresponding graph has minimum degree 3.

Lemma 3.2.16 *If there is any counterexample to Theorem 3.2.10, then there is a counterexample where* $\deg_G(u) \geq 3$ *for all* $u \in W$.

Proof: The proof is deferred to the Appendix at the end of this chapter. □

Proof (of Theorem 3.2.10): We can assume that if there is a counterexample $G = (B \dot\cup W, E)$ then G is connected, but the black subgraph $H := G[B]$ might not be connected. Moreover, by Lemma 3.2.16, we may assume that $\deg_G(u) \geq 3$ for all $u \in W$. Recall the notation from the previous subsection. We will show that there must exist a black vertex of degree at most 7, by establishing criterion (3.9) for G: If c_H denotes the number of components of H, by induction on c_H, it is easy to see that $c_H - 1 = \sum_{F \in \mathcal{F}} c_F$. Also, if z is the number of edges needed to triangulate G, we get $z = \sum_{F \in \mathcal{F}} z_F$. Hence—using the notation from Lemma 3.2.12—, we have

$$\begin{aligned} & 3w - z - c_H + f_H - 4b \\ = & \sum_{F \in \mathcal{F}} (3w_F + c_F - z_F + 1) - 2c_H + 1 - 4b \\ \leq & \sum_{F \in \mathcal{F}} (2t_F + 2) - 2c_H + 1 - 4b, \end{aligned}$$

here we used Propositions 3.2.14 and 3.2.15 in the last step.

Noting that $\sum_{F \in \mathcal{F}} t_F$ is the number of edges needed to triangulate H, we have

$$\sum_{F \in \mathcal{F}} t_F = 3b - 6 - m_{bb}.$$

The number of faces of H is $\sum_{F \in \mathcal{F}} 1 = f_H = m_{bb} - b + 1 + c_H$, by Euler's formula (3.13). Together, the criterion evaluates as follows

$$3w - z + -c_H + f_H - 4b \ \leq \ 2(3b - 6 - m_{bb}) + 2(m_{bb} - b + 1 + c_H) - 2c_H + 1 - 4b \ = \ -9.$$

By Lemma 3.2.12, this guarantees the existence of a vertex $v \in B$ with $\deg_G(v) \leq 7$. This is a contradiction to the assumption that G is a counterexample. □

3.2.4 Optimality of the Branching Theorem

We conclude this section by the observation that, with respect to the set of reduction rules we introduced in Subsection 3.2.2, the upper bound in our branching theorem is optimal. More precisely, there exists a planar reduced black and white graph with the property that all black vertices have degree 7. Such a graph is shown in Fig. 3.4. Moreover, this example can be generalized towards an infinite set of plane reduced black and white graphs with the property

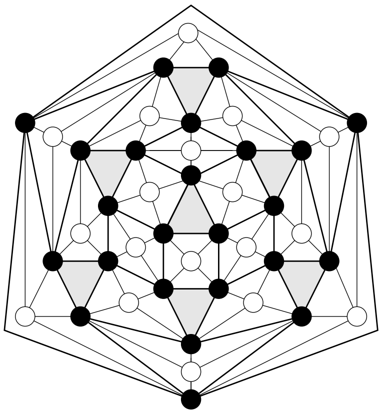

Figure 3.4: A graph that shows optimality of the bound derived in our branching theorem.

that all black vertices have degree 7. The example given in Figure 3.4 is the smallest of all graphs in this class. Let us describe this class of sample graphs in the following in more details. Each of the graphs could be imagined to be drawn on a can or, mathematically speaking, on a cylinder. On the bottom and the top of the cylinder, we embed the graph depicted in Fig. 3.5 (left-hand diagram). The vertices with numbers 1 through 9 are at the rim of the top or of the bottom of the can. These numbers are meant as an "interface" to the surface wrapped around the side face of the can. The (general) graph pattern used on the side face is depicted in Fig. 3.5 (right-hand diagram). It consists of two types of horizontal stripes. If the upper one is denoted by $S_{\square}$ and the lower one by $S_{\triangle}$, then consider some sidewall with the pattern described by the expression $S_{\triangle}(S_{\square}S_{\triangle})^n$ for some $n \geq 0$. Hereby, the upper row of black vertices in the uppermost stripe and the lower row of black vertices in the lowermost stripe of the type $S_{\triangle}$ are numbered $1, 2, 3, 4, 5, 6, 7, 8, 9, 1$. Identifying these vertices with the vertices on the rim of the top and the bottom of the can, respectively, we obtain the "can graph" G_n. The graph G_n has

$$2 \cdot 9 \cdot n \text{ [the side wall] } + 2 \cdot 12 \text{ [the top and bottom] } = 18n + 24$$

black vertices (each of degree seven) and

$$15 \cdot n + 6 \text{ [the side wall] } + 2 \cdot 6 \text{ [the top and bottom] } = 15n + 18$$

white vertices (each of degree four). As the reader may verify, G_0 is the graph depicted in Fig. 3.4. Moreover, none of the graphs G_n is reducible by means of any of the rules listed in Subsection 3.2.2.

It is an interesting and challenging task to ask for further reduction rules that would yield a provably better constant in the branching theorem. For example, one might think of the following straightforward generalization of bw-Rule 3.2:

bw-Rule 5 *If there is a white vertex* u *with the property that* $N_G(u) \subseteq N_G(v)$ *for some other (black or white) vertex, then delete* u*.*[9]

However, the graph in Fig. 3.4 is reduced even with respect to this bw-Rule 5.

[9]Note that it is not clear how to carry out this reduction rule in linear time.

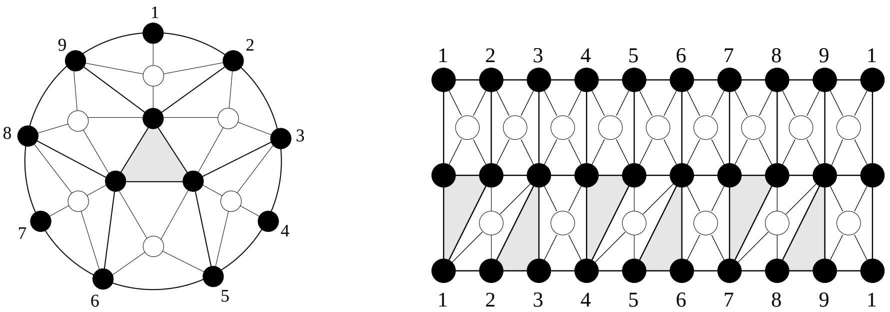

Figure 3.5: The left-hand diagram shows the top and bottom of the sample can. The right-hand diagram illustrates the sidewall pattern of the sample can.

Appendix

Proof of Lemma 3.1.6:
It is well-known that every planar graph contains a vertex of degree bounded at most 5 (see Section 1.2). For $g > 0$, let $G = (V, E) \in \mathbb{G}(\mathcal{S}_g)$, and let $n = |V|$ and $m = |E|$. If $n \le d_g + 1$, then clearly $\deg_G(u) \le d_g$ for all $u \in V$. Hence, we can assume

$$n \ge d_g + 2. \tag{3.17}$$

Suppose we had $\deg_G(u) > d_g$ for all $u \in V$. Then, by Euler's formula (see Eq. (1.3)), we get

$$d_g \cdot n < 2m \le 2(3n - 6 + 6g) = 6n + 12(g - 1),$$

which is equivalent to

$$(d_g - 6)n < 12(g - 1). \tag{3.18}$$

Hence, combining (3.17) and (3.18), and using that $g > 0$, we have

$$(d_g - 6)(d_g + 2) \le (d_g - 6)n < 12(g - 1)$$

from which we conclude that $d_g < 2(1 + \sqrt{3g + 1})$, a contradiction. □

Proof of Lemma 3.2.13:
We construct a tree connecting the V_1-vertices among themselves by recursively decrementing the number of components in $G[V_1]$ from $|V_1|$ to 1 by adding edges. This means that we are going to prove the lemma by induction over the number of components of $G[V_1]$. The induction base—where the number of these components equals one—trivially holds. In the induction step, we use the following claim.

Claim: Let $G = (V_1 \dot\cup V_2, E)$ be a plane graph, where V_1 is an independent set in G and where $G[V_2]$ is a forest. Then, for every vertex $v \in V_1$, there exists another vertex $v' \in V_1$ such that the edge $\{v, v'\}$ can be additionally drawn in the embedded graph G without destroying planarity.

Assume that the claim has been verified and that the assertion of the lemma holds for all graphs where $G[V_1]$ is a forest with c components. Consider now a graph G which satisfies the assumptions of this lemma and where $G[V_1]$ is a forest with $c+1$ components. Let the graph $G' = (V_1' \dot\cup V_2, E')$ be obtained from G by contracting all components of $G[V_1]$ to single vertices. Then, G' satisfies the assumption of the claim. Hence, a vertex can be drawn connecting two vertices u and u' in V_1' which represent components K and K' in G. Clearly, the edge e obtained by the claim can be drawn between two arbitrary vertices v and v' belonging to components K and K', respectively. Now, the induction hypothesis can be applied to $\hat{G} = G + e$, since $\hat{G}$ has only c components.

Proof of the Claim. Take some vertex $v \in V_1$. If there is no cycle enclosing v, it is possible to connect v with any other vertex in V_1 without destroying planarity. Otherwise, consider the set of all embedded cycles which enclose v. This set is partially ordered by the relation "cycle C_1 contains cycle C_2." Take the smallest of these cycles. Since $G[V_2]$ is acyclic by assumption, this cycle must contain at least one vertex v' from V_1. By construction, an edge can be drawn between v and v' without destroying planarity. □

Proof of Proposition 3.2.15:
We first of all proof the claim assuming that $\deg(u) = 3$ for all $u \in W$.

We use the same idea as in the proof for the special case where $G[B]$ was assumed to be connected. Let F be a face of the black induced subgraph $G[B]$. Let $W_F \subseteq W$ be the set of white vertices in the interior of F, and let $B_F \subseteq B$ denote the black vertices on the boundary of F.

For a subset $W' \subseteq W_F$, we again consider the bipartite graph

$$H(W') = (W' \dot\cup T(W'), E(W'))$$

as constructed in (3.16). It is again helpful to consult Fig. 3.3 for an example of this construction.

Instead of trying to find a W_F-saturated matching for $H(W_F)$ (as it was done in the special case where $G[B]$ was assumed to be connected), we now follow a different approach.

We already observed that the set $T(W')$ gives us vertices in $H(W')$ that correspond to pairs $e = \{b_1, b_2\}$ of black vertices in B_F between which we still can draw an edge in $G[B]$. The edge e can even be drawn in the interior of F, since b_1 and b_2 are connected by a white vertex in $W' \subseteq W_F$. In particular, this means that

$$|T(W_F)| \leq t_F. \tag{3.19}$$

Observe that, due to bw-Rule 4, for each $u \in W_F$, the neighbors $N(u) \subseteq B_F$ are connected by at most one edge in $G[B]$. By construction of $H(W_F)$, this means that

$$\deg_{H(W_F)}(u) \geq 2 \quad \text{for all } u \in W_F. \tag{3.20}$$

By construction, the degree $\deg_{H(W_F)}(e)$ for an element $e = \{b_1, b_2\} \in T(W_F)$ tells us how many white vertices share the pair $\{b_1, b_2\}$ as common neighbors. We do case analysis according to this degree.

Case 1: Suppose that $\deg_{H(W_F)}(e) \leq 2$ for all $e \in T(W_F)$. Then, $H(W_F)$ is a bipartite graph, in which the first bipartition set has degree at least two (see Eq. (3.20)) and the second bipartition

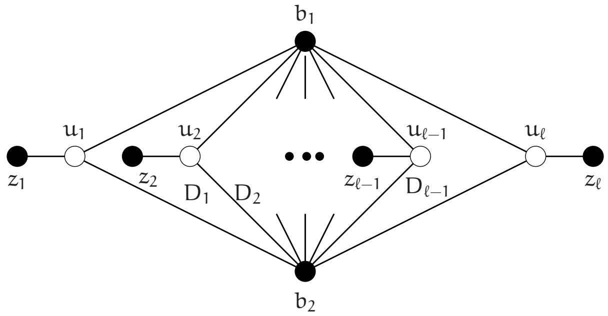

Figure 3.6: Illustration of a diamond D generated by a pair vertices $\{b_1, b_2\} \in T(W_F)$.

set has degree at most two. In this way, the second set cannot be smaller, which yields

$$w_F = |W_F| \le |T(W_F)| \overset{(3.19)}{\le} t_F.$$

Case 2: There exist elements $e = \{b_1, b_2\}$ in $T(W_F)$ which are shared as common neighbors by more than 2 white vertices (i.e., $\deg_{H(W_F)}(e) = \ell > 2$). Suppose that we have $u_1, \dots, u_\ell \in W_F$ with $N_G(u_i) = \{b_1, b_2, z_i\}$ (i.e., $\{u_i, e\} \in E(W_F)$). We may assume that the vertices are ordered such that the closed region D bounded by $\{b_1, u_1, b_2, u_\ell\}$ contains all other vertices $u_2, \dots, u_{\ell-1}$ (see Fig. 3.6).

We call D the *diamond* generated by $\{b_1, b_2\}$. Note that D consists of $\ell - 1$ regions, which we call *blocks* in the following; the block D_i is bounded by $\{b_1, u_i, b_2, u_{i+1}\}$ $(1 \le i \le \ell - 1)$. Let $W_i \subseteq W_F$ and $B_i \subseteq B_F$, respectively, denote the white and black vertices, respectively, that lie in D_i. For the boundary vertices $\{b_1, b_2, u_1, \dots, u_\ell\}$, we use the following convention: b_1, b_2 are added to all blocks, i.e., $b_1, b_2 \in B_i$ for all i; and u_i is added to the region where its third neighbor z_i lies in. A block is called *empty* if $B_i = \{b_1, b_2\}$ and, hence, $W_i = \emptyset$. Moreover, let $W_D := \bigcup_{i=1}^{\ell-1} W_i$ and $B_D := \bigcup_{i=1}^{\ell-1} B_i$.

We only consider diamonds where z_1 and z_ℓ are not contained in D (see Fig. 3.6). The other cases can be treated with similar arguments.

Note that each block of a diamond D may contain further diamonds, the blocks of which may contain further diamonds, and so on. Since no diamonds overlap, the topological inclusion forms a natural ordering on the set of diamonds and their blocks.

We now use the following claim.

Claim: For each diamond D generated by $\{b_1, b_2\}$, let t_D denote the number of black-black edges that can be added to $G[B]$ other than $\{b_1, b_2\}$. We claim that $t_D \ge |W_D|$ and that all of these t_D edges can be drawn inside D, in a way that we still have the possibility to draw the edge $\{b_1, b_2\}$ without any edge-crossing.

Using this claim, we can finish the proof: Consider all diamonds $D^1, \dots, D^r$ which are not contained in any further diamond. Suppose D^i has boundary $\{b_1^i, u_1^i, b_2^i, u_{\ell_i}^i\}$ with $b_1^i, b_2^i \in B_F$

and $u_1^i, u_{\ell_i}^i \in W_F$. Let

$$W_F' := W_F \setminus (\bigcup_{i=1}^{r} W_{D^i}).$$

According to the claim, we already found $\sum_{i=1}^{r} t_{D^i}$ many black-black edges in E^{poss} inside the diamonds D^i. Observe that each pair $e^i = \{b_1^i, b_2^i\}$ is only shared as common neighbors by at most two white vertices (namely, u_1^i and $u_{\ell_i}^i$) in (sic!) W_F'. Hence, the bipartite graph $H(W_F')$ again has the property that

- $\deg_{H(W_F')}(e) \leq 2$ for all $e \in T(W_F')$, and
- $\deg_{H(W_F')}(u) \geq 2$ for all $u \in W_F'$.[10]

Similar to Case 1 this proves that—additionally—we find t' (with $t' \geq |W_F'|$) many edges in E^{poss}. Hence,

$$w_F \;=\; |W_F| \;= |W_F'| + \left|\bigcup_{i=1}^{r} W_{D^i}\right| \;\leq\; t' + (\sum_{i=1}^{r} t_{D^i}) \;\leq\; t_F.$$

This finishes the proof under the assumption that $\deg(u) = 3$ for all $u \in W_F$.

Proof of the Claim. We give an inductive argument proceeding from the "innermost" diamonds to the outer ones with respect to the inclusion ordering mentioned above.

Induction base: Consider an innermost diamond D with its blocks $D_1, \ldots, D_{\ell-1}$. We give a proof for the claim in the case where z_1 and z_ℓ are not contained in D (see Fig. 3.6). The other cases work similarly. Suppose that there are $\ell' \leq \ell - 1$ many non-empty blocks. For each non-empty block, we consider the bipartite graph $H(W_i)$. Since D_i has no further diamonds in its interior, we again have the property that $\deg_{H(W_i)}(e) \leq 2$ for all $e \in T(W_i)$. This shows that $|W_i| \leq |T(W_i)|$ (with the same arguments as in Case 1). This means, that we can draw in each block at least $|W_i|$ many black black edges. Note that all edges $e \in T(W_i)$ can be drawn in the interior of D_i. This sums up to at least $\sum_{i=1}^{\ell-1} |W_i| = |W_D|$ many black black edges that can be drawn inside D as stated in the claim. There is one minor subtlety we have to be aware of: We might have used $\{b_1, b_2\}$ for each non-empty D_i, i.e., at most ℓ' times. Since (according to the claim) we do not wish to use the edge $\{b_1, b_2\}$ at all, we use a set of ℓ' many additional black black edges from E^{poss} instead. These additional edges can be found by the following observation: Consider the vertex z_i $(2 \leq i \leq \ell - 1)$. Suppose $z_i \in B_i$ (the case $z_i \in B_{i-1}$ works similarly). We distinguish the two cases that either D_{i-1} is an empty block or D_{i-1} is non-empty. If D_{i-1} is empty, then $z_{i-1} \in B_{i-2}$ and it is easy to see that a black-black edge $\{z_i, z_{i-1}\}$ (from D_i to D_{i-2}) could be additionally added. If D_{i-1} is a non-empty block, then it is not hard to see that there must be a vertex $x \in B_{i-1}$ (possibly z_{i-1}) so that we can add the additional black-black edge $\{z_i, x\}$ (from D_i to D_{i-1}). An easy analysis shows that this gives ℓ' many additional edges.

Induction step: Consider a diamond D generated by $\{b_1, b_2\}$ with blocks $D_1, \ldots, D_\ell$ and suppose that, for all further diamonds inside the blocks D_i, the claim already holds true. Suppose we

[10] Note that according to the claim the edges $\{b_1^i, b_2^i\}$ still can be used.

had "inner diamonds" $D_i^1, \ldots, D_i^{j_i}$ inside D_i. The induction hypothesis already assures that we find at least $\sum_{s=1}^{j_i} |W_{D_i^s}|$ many black-black edges from E^{poss} inside the diamonds $D_i^1, \ldots D_i^{j_i}$. Hence, it remains to consider $W_i' := W_i \setminus (\bigcup_{s=1}^{j_i} W_{D_i^s})$. The graph $H(W_i')$ has the properties that

- $\deg_{H(W_i')}(u) \geq 2$ for all $u \in W_i'$, and
- $\deg_{H(W_F')}(e) \leq 2$ for all $e \in T(W_i')$.

This means that we can argue similar to the induction base to see that we can find at least $\sum_{i=1}^{\ell} |W_i'|$ many additional black-black edges inside D not using the edge $\{b_1, b_2\}$. In total this gives us at least

$$\sum_{i=1}^{\ell} \left(|W_i'| + \sum_{s=1}^{j_i} |W_{D_i^s}| \right) = |W_D|$$

many edges. This finishes the proof of the claim.

We show in the following that the assumption that $\deg_G(u) = 3$ for all $u \in W_F$ is no restriction.

Observation 3.2.17 *If* F_1 *and* F_2 *are two faces of* $G[B]$ *with common boundary edge* e*, then* $t_{F_1} + t_{F_2} + 1$ *equals* t_F*, where we now consider* $(G - e)[B]$*, and* F *is the face which results from merging* F_1 *and* F_2 *when deleting* e*.*

Consider a black and white graph $G = (B \dot{\cup} W, E)$ that is reduced with respect to bw-Rule 4 and for which $\deg(u) \geq 3$ holds for all $u \in W$. If there is some $u \in W$ with $\deg(u) > 4$, then delete arbitrarily all edges incident with u but four of them. While preserving the black induced subgraph, the resulting graph is still reduced with respect to bw-Rule 4, since this rule does not apply to white degree-4-vertices. Therefore, we can assume from now on w.l.o.g. that all white vertices of G have maximum degree of four.

We will now show the claim by induction on the number w^4 of white vertices of degree four. The above proof for the case $\deg(u) = 3$ for all $u \in W$ can be taken as induction base. Assume that the claim was shown for each graph with $w^4 \leq \ell$ and consider now the case that G has $\ell + 1$ white degree-4-vertices. Choose some arbitrary $u \in W$ with $\deg(u) = 4$. Let $\{b_1, \ldots, b_4\}$ be the clockwisely ordered neighbors of u. Due to planarity, we may assume further that $\{b_1, b_3\} \notin E$ without loss of generality. Consider now $G' = (G - u) + \{b_1, b_3\}$. We prove below that G' (or $G'' = (G - u) + \{b_2, b_4\}$ in one special case) is reduced with respect to bw-Rule 4. This means that the induction hypothesis applies to G'. Hence, $w_F \leq t_F$ for all faces in $G'[B]$. Observe that G' contains all the faces of G except for the face F of G which contains u; F might be replaced by two faces F_1 and F_2 with common boundary edge $\{b_1, b_3\}$. In this case, $w_{F_1} \leq t_{F_1}$, $w_{F_2} \leq t_{F_2}$, $w_{F_1} + w_{F_2} + 1 = w_F$ and, by Observation 3.2.17, $t_{F_1} + t_{F_2} + 1 = t_F$. Hence, $w_F \leq t_F$ by induction. In the case where face F still exists in G', it is trivial to see that $w_F \leq t_F$.

To complete the proof, we argue why G' has to be reduced with respect to bw-Rule 4. Obviously, this is clear if $\forall b_i \forall v \in N(b_i) : \deg(v) = 4$, since bw-Rule 4 only applies to degree-3-vertices. We now discuss the case that u has degree-3-vertices as neighbors.

(i) If a degree-3-vertex v is neighbor of some b_i, but not of b_j, $j \neq i$, then bw-Rule 4 will not apply to v in G', if it has not been applicable in to v in G already.

(ii) Consider the case that a degree-3-vertex is neighbor v of two b_i, b_j, $i \neq j$. If $|\{i,j\} \cap \{1,3\}| \leq 1$, then introducing the edge $\{b_1, b_3\}$ will not add any further edge to $N(v)$. Hence, bw-Rule 4 will not be applicable to v in G' unless we could have applied this rule already in G. If $\{i,j\} = \{1,3\}$, then, by planarity, $\{b_2, b_4\} \notin E(G)$ and we could consider $G'' = (G - u) + \{b_2, b_4\}$ instead of G' with an argument similar to the case $\{i,j\} = \{2,4\}$.

(iii) If a degree-3-vertex is neighbor of three b_i, b_j, b_k, then a reasoning similar to the one in the previous point applies.

This concludes the full proof of the proposition. □

Proof of Lemma 3.2.16:
Suppose G is a counterexample to the theorem. Then, G does not have any white vertices of degree one, else bw-Rule 2 can be applied. Let G' be obtained from G by simultaneously replacing every white vertex u of degree two with neighbors x and y by an edge $\{x, y\}$. The neighbors x and y of u are necessarily black, else bw-Rule 1 can be applied, and in each case the edge $\{x, y\}$ is not already present in G else bw-Rule 3.1 would apply. We argue that G' is reduced. If not, then the only possibility is that bw-Rule 4 applies to some white vertex u' of degree three in G'. If bw-Rule 4 did not apply to u' in G, then one of the edges between the neighbors of u' must have been created in our derivation of G' from G, i.e., one of these edges replaced a white vertex u of degree two. But this implies that bw-Rule 3.2 could be applied in G to u', contradicting that G is reduced. □

Chapter 4

Graph Separation

Often a problem can be solved by breaking it into smaller subproblems of similar kind, recursively solving these subproblems and then combining the solutions of the subproblem to get a solution for the original problem. This strategy is well-known as the divide-and-conquer method. The key ingredient to make a divide-and-conquer approach applicable to graph problems is the notion of (vertex) separators.

In this chapter, we will pursue a divide-and-conquer approach based on graph separation in combination with problem kernel reduction (see Chapter 2) in order to design fixed-parameter algorithms for various graph problems. The first subsection discusses the general concept. There, we focus on a general (mathematically precise) formulation of divide-and-conquer using graph separation. In addition, we coin the notion of "glueable vertex selection problems" to characterize the problems that allow for such a divide-and-conquer approach.

The three main contributions of this chapter are the following: Firstly, in Section 4.2, we consider planar graph problems. We show that various parameterized problems, including our threesome par-VERTEX COVER, par-INDEPENDENT SET, and par-DOMINATING SET, allow for "sublinear-exponential algorithms" running in time $2^{O(\sqrt{k})} + n^{O(1)}$. To our knowledge, these are the first algorithms with a sublinear term in the exponent of the running time. Moreover, we will see that these algorithms are asymptotically optimal. More precisely, we will show that an algorithm of running time $2^{o(\sqrt{k})} + n^{O(1)}$ for any of the three aforementioned problems would imply that 3 SAT $\in$ DTIME$(2^{o(n)})$, where n is the number of variables. This is considered to be unlikely in classical complexity theory.

Secondly, in Section 4.3, we take a glance beyond planar graphs and discuss disk (intersection) graphs of bounded radius ratio. Here, separator theorems are only known for the case of so-called ϑ-precision disk graphs (see Subsection 4.3.3). We prove a new geometric separator theorem that does not need this extra assumption and holds for all disk graphs of bounded radius ratio.

Finally, in Subsection 4.3.4, this new separator theorem then is used to solve the par-INDEPENDENT SET problem on disk graphs of bounded radius ratio. The problem is motivated from frequency assignment problems in cellular networks. We give an algorithm that runs in time $2^{O(\sqrt{k}\log(n))}$, a running time that is unlikely to be achievable for general graphs [119]. More precisely, if INDEPENDENT SET can be solved in time $2^{o(n)}$ then 3 SAT $\in$ DTIME$(2^{o(n)})$, where (in the latter case) n is the number of variables in a 3 SAT formula. Moreover, for the case of ϑ-precision disk graphs, we obtain fixed-parameter algorithms with a sublinear term in the exponent (i.e., running in time $2^{O(\sqrt{k})} + n^{O(1)}$) for various problems including par-VERTEX COVER, par-INDEPENDENT SET, and par-DOMINATING SET.

4.1 Background

This first introductory section is meant to familiarize the reader with known separator theorems and to provide a mathematical sound framework for the use of separator theorems for algorithmic purposes. We will describe, using several examples, how certain graph problems can be solved by a divide-and-conquer approach based on graph separation. The goal here is to point to various difficulties that arise for structurally sophisticated problems.

4.1.1 Classical $\sqrt{\cdot}$-Separator Theorems

We briefly discuss (classical) vertex separator theorems. For later purposes, we introduce a somewhat generalized notion of separator theorems.

Definition 4.1.1 *A* separator $V_S \subseteq V$ *of a graph* $G = (V, E)$ *partitions* V *into two* parts V_A *and* V_B *such that*

- $V_A \dot\cup V_S \dot\cup V_B = V$, *and*
- *no edge joins a vertex of* V_A *to* V_B.

The triple (V_A, V_S, V_B) *is also called a* separation *of* G. *In the following we sometimes use the notion* $\delta V_A := V_A \cup V_S$ *and* $\delta V_B := V_B \cup V_S$.

In order to provide a quantitative approach to separators, we need the notion of "measure."

Definition 4.1.2 *Let* $\mathbb{G}$ *be a graph class closed under taking subgraphs. A function* $\xi : \mathbb{G} \to \mathbb{R}^+$ *that is monotonous with respect to the subgraph ordering, i.e.,* $\xi(G) \leq \xi(G')$ *if* $G \subseteq G'$, *and for which* $\xi((\emptyset, \emptyset)) = 0$, *is called a* graph measure.

Example 4.1.3 The usual *counting measure* $|\cdot|$ which assigns to a graph G the size of its vertex set $|V_G|$ is clearly a graph measure. Later in this chapter we will use other graph measures for disk intersection graphs.

Definition 4.1.4 *Let* ξ *be a graph measure. An* $f(\cdot)$-vertex separator theorem for the measure ξ (and constants $\alpha < 1$, $\beta > 0$) *on a class of graphs* $\mathbb{G}$ *which is closed under taking subgraphs is a theorem of the following form:*
For any $G \in \mathbb{G}$ *there exists a separation* (V_A, V_S, V_B) *of* G *such that*

(i) $\xi(G[V_S]) \leq \beta \cdot f(\xi(G))$, *and*

(ii) $\xi(G[V_A]), \xi(G[V_B]) \leq \alpha \cdot \xi(G)$.

β	$\alpha = \frac{2}{3}$		$r(\frac{2}{3}, \beta)$	$\alpha = \frac{1}{2}$		$r(\frac{1}{2}, \beta)$	$\alpha = \frac{3}{4}$		$r(\frac{3}{4}, \beta)$
upper bounds	$\sqrt{\frac{2}{3}} + \sqrt{\frac{4}{3}}$	[77]	10.74	$\sqrt{24}$	[17, 45]	16.73	$\sqrt{\frac{2\pi}{\sqrt{3}}} \cdot \frac{1+\sqrt{3}}{\sqrt{8}}$	[179]	13.73
lower bounds	1.55	[75]	8.45	1.65	[179]	5.63	1.42	[179]	10.60

Table 4.1: Summary of various $\sqrt{\cdot}$-separator theorems on planar graphs with their constants α and β. Here, $r(\alpha, \beta)$ denotes the ratio $r(\alpha, \beta) = \beta/(1-\sqrt{\alpha})$, which is of central importance to the running time analysis of our algorithms (see Proposition 4.1.15).

$\sqrt{\cdot}$-separator theorems on planar graphs. Stated in this framework, the planar separator theorem due to Lipton and Tarjan [137] can be formulated as follows.

Theorem 4.1.5 *On the class of planar graphs, there exists a $\sqrt{\cdot}$-separator theorem for the counting measure $|\cdot|$ with constants $\alpha = \frac{2}{3}$ and $\beta = 2\sqrt{2}$. Moreover, the corresponding separation can be found in linear time.*

Later, Djidjev [75] showed an improved planar $\sqrt{\cdot}$-separator theorem with constants $\alpha = 2/3$ and $\beta = \sqrt{6} \approx 2.45$, which was further improved to $\alpha = 2/3$ and $\beta = \sqrt{4.5} \approx 2.12$ by Alon *et al.* [20]. The current record for $\alpha = 2/3$ is $\beta = \sqrt{2/3} + \sqrt{4/3} \approx 1.97$ [77]. Djidjev has also shown a lower bound of $\beta \approx 1.55$ for $\alpha = 2/3$ [75]. For $\alpha = 1/2$, the "record" of $\beta = 7 + 1/\sqrt{3} \approx 7.58$ due to Venkatesan [187] was recently outperformed by Bodlaender [45], yielding $\beta = 2\sqrt{6} \approx 4.90$. A lower bound of $\beta \approx 1.65$ is known in this case [179]. For $\alpha = 3/4$, the best known value for β is $\sqrt{2\pi/\sqrt{3}} \cdot (1+\sqrt{3})/\sqrt{8} \approx 1.84$ with a known lower bound of $\beta \approx 1.42$ (see [179]). The results are summarized in Table 4.1.

$\sqrt{\cdot}$-separator theorems on other graph classes. Similar to the case of planar graphs, $\sqrt{\cdot}$-separator theorems are also known for other graph classes, e.g., for the class of graphs of bounded genus, see [76]. More generally, Alon, Seymour, and Thomas proved a $\sqrt{\cdot}$-separator theorem for graph classes with an excluded complete graph minor [18, 19]. Besides, a $\sqrt{\cdot}$-separator theorem is known on the class of so-called "ℓ-map graphs," a generalized notion of planar graphs [57, 58]. Finally, we will also be concerned with graph separator theorems for so-called disk intersection graphs. This class of graphs will be treated separately in Section 4.3.

Conversely, finding separators is not possible in general graphs, as the example of the complete graph K_n with n vertices shows.

4.1.2 Algorithms Based on Graph Separation

From an algorithmic point of view, graph separators are important since they can be used to design divide-and-conquer strategies. We will give a characterization of a whole class of problems that can be attacked by such a divide-and-conquer strategy. To this end, in a first subsection, we coin the notions of "glueable vertex selection" graph problems. In a second subsection, we give a detailed analysis of a divide-and-conquer algorithm for such problems.

4.1.2.1 Glueability

As already Lipton and Tarjan [138] do, we are going to solve graph problems recursively, slicing the given graph into small pieces with the help of small separators.[1] Since we only consider *vertex* separators, we restrict ourselves to problems in which the optimal solutions consist of a subset of vertices.

Definition 4.1.6 *Let $\mathcal{G} = (\mathrm{I}_{\mathcal{G}}, \mathsf{F}_{\mathcal{G}}, \mathsf{c}_{\mathcal{G}}, \mathrm{opt}_{\mathcal{G}})$ be an optimization problem on graphs, i.e., $\mathrm{I}_{\mathcal{G}}$ is the set of graphs. We say that $\mathcal{G}$ is a* vertex selection problem *if the feasible solutions $\mathsf{F}_{\mathcal{G}}(\mathsf{G})$ for a graph $\mathsf{G} \in \mathrm{I}_{\mathcal{G}}$ are subsets of the vertices of $\mathsf{V}(\mathsf{G})$.*

Note that all problems we investigate in this work are vertex selection problems.

We begin with a brief discussion of divide-and-conquer algorithms based on separation. As a first example, we consider the VERTEX COVER problem for which a straightforward algorithm will work. We then turn our attention to the (much more) elusive DOMINATING SET problem in order to get insight into the difficulties that we encounter when trying to carry over the strategy used for VERTEX COVER. This will then lead to a general characterization of problems that allow for a separation based divide-and-conquer approach.

Suppose now, we wish to compute an optimal vertex cover of a graph G that is given together with a separation $(\mathsf{V}_\mathsf{A}, \mathsf{V}_\mathsf{S}, \mathsf{V}_\mathsf{B})$. The general scheme to obtain a solution can be sketched as follows:

- Find a separation $(\mathsf{V}_\mathsf{A}, \mathsf{V}_\mathsf{S}, \mathsf{V}_\mathsf{B})$ of the current graph.
- Consider all possible 0-1-assignments ζ of the vertices in V_S (where assigning "color" 1 to a vertex means to choose the vertex to belong to the vertex cover and "color" 0 means not to choose the vertex). For each such assignment ζ,
  - check whether the 0-1-assignment ζ leads to a feasible solution in $\mathsf{G}[\mathsf{V}_\mathsf{S}]$,
  - divide the problem into two subproblems on some subgraphs $\mathsf{G}_\mathsf{A}^\zeta \subseteq \mathsf{G}[\delta\mathsf{V}_\mathsf{A}]$ and $\mathsf{G}_\mathsf{B}^\zeta \subseteq \mathsf{G}[\delta\mathsf{V}_\mathsf{B}]$,
  - recursively compute the optimal vertex cover on $\mathsf{G}_\mathsf{A}^\zeta$ and $\mathsf{G}_\mathsf{B}^\zeta$, and
  - combine the solutions to obtain a solution for G for the current assignment ζ.
- Return the best solution among all choices of ζ.

More precisely, when dealing with a specific 0-1-assignment ζ for V_S, then, for $\mathsf{X} \in \{\mathsf{A}, \mathsf{B}\}$, all vertices $\mathsf{V}_\mathsf{X}^\zeta := \{v \in \mathsf{V}_\mathsf{X} \mid \exists w \in \mathsf{V}_\mathsf{S} \cap \mathsf{N}(v), \zeta(w) = 0\}$ need to belong to a feasible vertex cover for G under the current assignment ζ. This is clear since each edge $\{v, w\}$ between vertices in V_A and in V_S needs to be covered. Taking these vertices to the solution it remains to find an optimal solution for the subgraphs $\mathsf{G}_\mathsf{X}^\zeta := \mathsf{G}[\mathsf{V}_\mathsf{X} \setminus \mathsf{V}_\mathsf{X}^\zeta]$. We let $\mathrm{vc}(\mathsf{G}[\mathsf{V}_\mathsf{S}], \zeta)$ denote the number of vertices

[1] Lipton and Tarjan only describe in details a solution for the structurally simple INDEPENDENT SET problem. In contrast, we also deal with more elusive problems here, such as DOMINATING SET.

that are assigned color 1 by ζ, if ζ leads to a feasible vertex cover in $G[V_S]$, and $\mathrm{vc}(G[V_S], \zeta) = \infty$, otherwise. The correctness of the divide-and-conquer approach is established by the equation

$$\mathrm{vc}(G) = \min_{\zeta: V_S \to \{0,1\}} \Big\{ \mathrm{vc}(G[V_S], \zeta) + \sum_{X \in \{A,B\}} (\nu c(G_X^{\zeta}) + |V_X^{\zeta}|) \Big\},$$

which is easily seen to hold true by the above given arguments.

When trying to carry over this scheme to the more involved DOMINATING SET problem, we face two immediate difficulties. Firstly, it is *not* sufficient to assign two colors to the vertices in V_S: If some vertex $v \in V_S$ is assigned color 0 (i.e, v is not chosen to belong to the dominating set), v needs to be dominated by some other vertex. There are three possibilities of doing so. Either, v is dominated by a vertex in V_S, or by a vertex in V_A, or by a vertex in V_B. In this sense, a more sophisticated color assignment is necessary to express these possibilities. Here, we could replace the single color 0 by the color set $\{0_A, 0_S, 0_B\}$ with the semantics that 0_X asks for a domination from the set V_X with $X \in \{A, S, B\}$. Hence, we have to deal with color assignments $\zeta : V_S \to \{1, 0_A, 0_S, 0_B\}$. Secondly, when dividing the problem into two subproblems, both, the information that vertices in $\zeta^{-1}(\{0_X\})$ need to be dominated by some vertex in V_X and the information that vertices in $\zeta^{-1}(\{1\})$ already might dominate vertices in V_X, must be handed down to the recursive calls. Hence, for $X \in \{A, B\}$, the graph G_X^{ζ} that we use in the recursive call should bear all this necessary information. We could let $G_X^{\zeta} := G[V_X \cup \zeta^{-1}(\{0_X, 1\})]$ and ask for a solution on G_X^{ζ} under the constraint that vertices in $\zeta^{-1}(\{1\})$ belong to the dominating set and all vertices in $\zeta^{-1}(\{0_X\})$ do not belong to the dominating set (and, hence, need to be dominated by some vertex in V_X). This forces us to find solutions on so-called "precolored graphs."

We now build a formal framework which enables us to handle both difficulties. In doing so, we follow parts of [9].

Definition 4.1.7 *A* (partial) 0-1-coloring *of a graph* G *is a function* $\xi : \mathrm{supp}(\xi) \to \{0, 1\}$ *which assigns colors to some subset* $\mathrm{supp}(\xi) \subseteq V(G)$*, which we call the* support *of* ξ*.*

A pair (G, ξ) *consisting of a graph* G *and a* 0-1*-coloring* ξ *is called a* precolored graph*.*

A vertex selection problem can be extended to precolored graphs in the following way.

Definition 4.1.8 *Let* $\mathcal{G} = (I_{\mathcal{G}}, F_{\mathcal{G}}, c_{\mathcal{G}}, \mathrm{opt}_{\mathcal{G}})$ *be a vertex selection problem. For a precolored graph* (G, ξ)*, we say that* $V' \in F_{\mathcal{G}}(G)$ *is* consistent with ξ*, if*

$$\forall v \in \mathrm{supp}(\xi) : \quad (\xi(v) = 0 \Rightarrow v \notin V') \ \wedge \ (\xi(v) = 1 \Rightarrow v \in V').$$

We then define

$$\mathrm{OPT}_{\mathcal{G}}(G, \xi) = \mathrm{opt}_{\mathcal{G}} \big\{ c_{\mathcal{G}}(V') \mid V' \in F_{\mathcal{G}}(G) \textit{ and } V' \textit{ is consistent with } \xi \big\}.$$

Informally speaking, the value $\mathrm{OPT}_{\mathcal{G}}(G, \xi)$ is the optimal solution of a vertex selection problem under the constraint that vertices which are assigned color 0 by the precoloring ξ must not belong to the solution and vertices which are assigned color 1 by the precoloring ξ must belong to the solution.

As discussed in the example of DOMINATING SET, it will be necessary to also allow larger color sets for the internal assignments to the vertices of a separator V_S in the divide-and-conquer algorithm: Let C_0, C_1 be finite, disjoint sets (called the *color sets*). Extending the notion of 0-1-colorings for a graph G, we call a function $\zeta : \mathrm{supp}(\zeta) \to C_0 \cup C_1$ with support $\mathrm{supp}(\zeta) \subseteq V(G)$ a C_0-C_1-*coloring*.

Definition 4.1.9 *Let ξ be a 0-1-coloring, and ζ be a C_0-C_1-coloring for a graph G. We say that ζ is* consistent with ξ *if*

$$\forall v \in \mathrm{supp}(\xi),\ i = 0,1 : \quad \xi(v) = i \Rightarrow \zeta(v) = C_i.$$

We now define so-called "glueable" vertex selection problems, a formal framework for those problems that can be solved with separator based divide-and-conquer techniques as described above. We apply the rather abstract notion to concrete graph problems afterwards (see Lemma 4.1.11). However, compared to [9], where this notion was coined, our definition here is significantly simplified.

Definition 4.1.10 *A vertex selection problem $\mathcal{G}$ is* glueable with λ colors *if there exists a color set $C := C_0 \dot\cup C_1$ with $|C| = \lambda$ such that, if we are given a precolored graph (G, ξ) together with a separation (V_A, V_S, V_B) of G, we find for each C_0-C_1-coloring $\zeta : V_S \to C$, precolored subgraphs*

- $(G_A^{\xi,\zeta}, \eta_A^{\xi,\zeta})$ *with* $G_A^{\xi,\zeta} \subseteq G[\delta V_A]$ *and some partial 0-1-coloring* $\eta_A^{\xi,\zeta}$ *of* $G_A^{\xi,\zeta}$,
- $(G_S^{\xi,\zeta}, \eta_S^{\xi,\zeta})$ *with* $G_S^{\xi,\zeta} \subseteq G[V_S]$ *and some partial 0-1-coloring* $\eta_S^{\xi,\zeta}$ *of* $G_S^{\xi,\zeta}$*, and*
- $(G_B^{\xi,\zeta}, \eta_B^{\xi,\zeta})$ *with* $G_B^{\xi,\zeta} \subseteq G[\delta V_B]$ *and some partial 0-1-coloring* $\eta_B^{\xi,\zeta}$ *of* $G_B^{\xi,\zeta}$,

such that

$$\mathrm{OPT}_{\mathcal{G}}(G,\xi) = \mathrm{opt}_{\mathcal{G}}\Big\{ \sum_{X \in \{A,S,B\}} f_X^{\xi,\zeta}(\,\mathrm{OPT}_{\mathcal{G}}(G_X^{\xi,\zeta}, \eta_X^{\xi,\zeta})\,) \ \Big| \ \zeta : V_S \to C,\ \ \zeta \text{ is consistent with } \xi \Big\} \tag{4.1}$$

for some polynomial time computable functions $f_X^{\xi,\zeta} : \mathbb{N} \cup \{\infty\} \to \mathbb{N} \cup \{\infty\}$.

We give some examples for glueable vertex selection problems.

Lemma 4.1.11 VERTEX COVER *and* INDEPENDENT SET *are glueable with 2 colors and* DOMINATING SET *is glueable with 4 colors.*

Proof: For the VERTEX COVER problem we choose the simple color sets $C_i := \{i\}$ $(i = 0, 1)$. Now, suppose we are given a precolored graph (G, ξ) together with a separation (V_A, V_S, V_B), and some coloring $\zeta : V_S \to \{0, 1\}$. Compare the following construction with the example given in Fig. 4.1.

Firstly, we let $(G_S^{\xi,\zeta}, \eta_S^{\xi,\zeta}) = (G[V_S], \zeta)$, i.e., $G[V_S]$ is fully colored. Then $\mathrm{OPT}(G[V_S], \zeta)$ is the number of vertices assigned color 1 if the coloring ζ is a valid vertex cover, otherwise $\mathrm{opt}(G[V_S], \zeta) = \infty$. Simply let $f_S^{\xi,\zeta}(x) = x$.

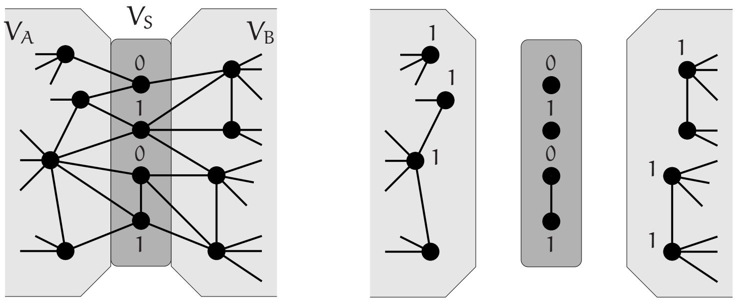

Figure 4.1: Glueability of VERTEX COVER with 2 colors (according to Definition 4.1.10). The left-hand diagram shows a separation of a graph with a 0-1-coloring ζ of V_S. The right hand diagram illustrates the three precolored subgraphs $(G_X^{\xi,\zeta}, \eta_X^{\xi,\zeta})$ with $X \in \{A, S, B\}$ into which the problem is subdivided.

Secondly, the precolored subgraphs $(G_X^{\xi,\zeta}, \eta_X^{\xi,\zeta})$ (with $X \in \{A, B\}$) are constructed in such a way that all edges going from V_X to V_S will be covered. Hence, let $G_X^{\xi,\zeta} := G[V_X]$ with a precoloring according to the following scheme (see Fig. 4.1): If there is an edge $\{v, w\}$ with $v \in V_X$ and $w \in V_S$, then v is assigned color 1, if vertex w is assigned color 0 by ζ. This forces us to cover the edge $\{v, w\}$ by v. Conversely, if w is assigned color 1 by ζ, the edge $\{v, w\}$ is already covered by w and we leave it open, whether v will belong to the vertex cover or not. By this construction, $\mathrm{OPT}(G[V_X], \eta_X^{\xi,\zeta})$ is the size of a minimal vertex cover of $G[V_X]$ under the assumption that all neighbors of vertices w in V_S with $\zeta(w) = 0$ will belong to the vertex cover. In addition, to keep the coloring consistent with ξ, every vertex $v \in V_X \cap \mathrm{supp}(\xi)$ will keep the color assigned by ξ. If it happens that a vertex in V_X that is assigned color 0 by ξ and at the same time has a neighbor in V_S being assigned color 0 by ζ, then we will not get a feasible solution, so, in this case we set, $f_X^{\xi,\zeta}(x) = \infty$. Otherwise let $f_X^{\xi,\zeta}(x) = x$.

It is clear that $\sum_{X\in\{A,S,B\}} f_X^{\xi,\zeta}(\mathrm{OPT}(G_X^{\xi,\zeta}, \eta_X^{\xi,\zeta}))$ then gives the size of a minimal vertex cover of the graph G precolored by ξ and ζ.

INDEPENDENT SET is shown to be glueable with 2 colors by a similar idea.

To show that DOMINATING SET is glueable with 4 colors, we use the color sets $C_1 := \{1\}$ and $C_0 := \{0_A, 0_S, 0_B\}$. The semantics of these colors were already discussed at the beginning of this subsection: Assigning the color 0_X, for $X \in \{A, S, B\}$, to vertices in V_S means that the vertex is not in the dominating set and will be dominated by a vertex in X. Clearly, 1 will mean that the vertex belongs to the dominating set. Suppose we are given a precolored graph (G, ξ) together with a separation (V_A, V_S, V_B) and let $\zeta : V_S \to C_0 \cup C_1$ be some C_0-C_1-coloring. For the following construction the example given in Fig. 4.2 is helpful.

Firstly, we let $(G_S^{\xi,\zeta}, \eta_S^{\xi,\zeta})$ be the graph $G[\zeta^{-1}(\{1, 0_S\})] \subseteq G[V_S]$ that is (fully) precolored with the coloring induced by ζ. Then, the value $\mathrm{OPT}(G_S^{\xi,\zeta}, \eta_S^{\xi,\zeta})$ tells us whether ζ is a feasible coloring in the sense that all vertices that are assigned color 0_S really are dominated by vertices in V_S. More precisely, if the coloring is feasible, then this value equals the number of vertices assigned color 1, and if this is not the case, this value will be ∞.

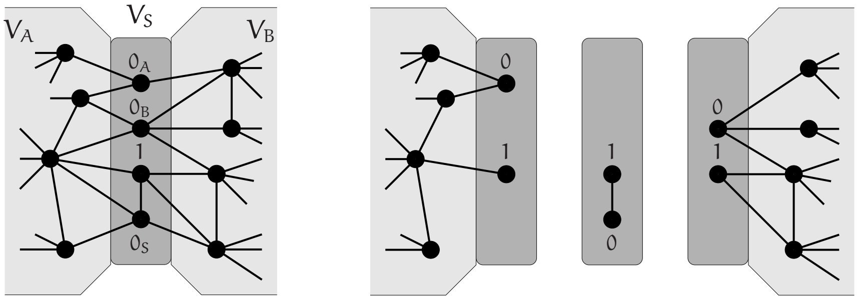

Figure 4.2: DOMINATING SET is glueable (according to Definition 4.1.10) with 4 colors $\{1, 0_A, 0_S, 0_B\}$. The left-hand diagram shows a separation of a graph together with a coloring ζ of V_S. The right hand diagram illustrates the three precolored subgraphs $(G_X^{\xi,\zeta}, \eta_X^{\xi,\zeta})$, $X \in \{A, S, B\}$, into which the problem is subdivided.

Secondly, the precolored subgraphs $(G_X^{\xi,\zeta}, \eta_X^{\xi,\zeta})$ (with $X \in \{A, B\}$) are constructed as follows. When dividing the problem into subproblems, we need to hand down the information whether a vertex in V_S is assigned color 1 by ζ (and, hence, its neighbors in V_X can be assumed to be dominated) and the information whether a vertex in V_S is assigned color 0_X by ζ (and, hence, one of its neighbors in V_X needs to dominate this vertex). Thus, we let $G_X^{\xi,\zeta} := G[V_X \cup \zeta^{-1}(\{1, 0_X\})]$ together with the obvious precoloring $\eta_X^{\xi,\zeta}$ that assigns color 1 to vertices in $\zeta^{-1}(\{1\})$ and color 0 to vertices in $\zeta^{-1}(\{0_X\})$. Moreover, to keep the coloring consistent with ξ, every vertex $v \in V_X \cap \operatorname{supp}(\xi)$, will keep the color assigned by ξ. The term $\mathrm{OPT}(G_X^{\xi,\zeta}, \eta_X^{\xi,\zeta})$ then computes (for each C_0-C_1-coloring ζ) the size of a minimum dominating set in $G[V_X]$ under the constraint that some vertices in δV_X still need to be dominated (namely, the vertices in $\delta V_X \cap \zeta^{-1}(\{0_X\})$) and some vertices in δV_X can already be assumed to be in the dominating set (namely, the vertices in $\delta V_X \cap \zeta^{-1}(\{1\})$).

Finally, to satisfy Equation (4.1), we let $f_S^{\xi,\zeta}(x) = x$ and, for $X \in \{A, B\}$, we let $f_X^{\xi,\zeta}(x) = x - |\zeta^{-1}(\{1\})|$. By this, we assure that vertices from V_S which are assumed to be in the dominating set are not counted multiple times. □

Remark 4.1.12 DOMINATING SET was shown to be glueable with $\lambda = 4$ colors. We remark that—for algorithmic purposes—it can be dropped to $\lambda = 3$ by following argument: Recall the color set $C = \{1, 0_A, 0_S, 0_B\}$ and its semantics from the proof of Lemma 4.1.11. Instead of assigning all colors from C to the vertices in the separator, we use only a color assignment $\zeta : V_S \to C'$ with the *three* colors $C' = \{1, 0_A, 0_B\}$. For each such coloring, we then "overwrite" the set

$$\{\, v \in V_S \mid \zeta(v) \in \{0_A, 0_A\} \text{ and } \exists w \in N(v) \cap V_S : \ \zeta(w) = 1 \,\}$$

with the color 0_S, since the vertices in this set are the ones that are already dominated by vertices in V_S.

Remark 4.1.13 We mention in passing that—besides the problems stated in the preceding Lemma 4.1.11—many more vertex selection problems are glueable. For example, weighted versions and variations of the problems discussed in Lemma 4.1.11, such as the par-DOMINATING

```
double compute_OPT (precolored graph (G, ξ))
/* input:   a precolored graph (G, ξ) instance of                        */
/*          a vertex selection problem 𝒢 = (I_𝒢, F_𝒢, c_𝒢, opt_𝒢).         */
/* output:  the solution OPT_𝒢((G, ξ)) as defined in Definiton 4.1.8.     */
```

- if (supp(ξ) $= V(G)$) then
  - return $c_{\mathcal{G}}(V')$ for $V' \subseteq V(G)$ that is consistent with ξ and exit.
- use a $\sqrt{\cdot}$-separator theorem to find a separation (V_A, V_S, V_B) for G.
- for all C_0-C_1-colorings $\zeta : V_S \to C_0 \dot\cup C_1$ that are consistent with ξ do
  - construct the three precolored graphs $(G_X^{\xi,\zeta}, \eta_X^{\xi,\zeta})$ with $X \in \{A, S, B\}$.
  - recursively compute $c_X^{\xi,\zeta} :=$ compute_OPT$(G_X^{\xi,\zeta}, \eta_X^{\xi,\zeta})$ with $X \in \{A, S, B\}$.
  - evaluate $c^{\xi,\zeta} := \sum_{X \in \{A,S,B\}} f_X^{\xi,\zeta}(c_X^{\xi,\zeta})$.
- return opt$\{ c^{\xi,\zeta} \mid \zeta : V_S \to C, \ \ \zeta$ is consistent with $\xi \}$.

Figure 4.3: Divide-and-conquer algorithm for a glueable vertex selection problem.

SET WITH PROPERTY P problems which are listed in Subsection 1.3.4 are glueable. As an example for a problem where more than one color in C_1 is needed, we highlight the TOTAL DOMINATING SET problem, for which we can use the color sets $C_i := \{i_A, i_S, i_B\}$ $(i = 0, 1)$ where the semantics of assigning color i_X $(X \in \{A, S, B\})$ to a vertex in a separator V_S is that this vertex either is taken in the dominating set $(i = 1)$ or not $(i = 0)$, and that we require to dominate the vertex by a neighbor in V_X.

4.1.2.2 Divide-and-Conquer

The notion of glueable vertex selection problems is tailored in a way such that divide-and-conquer approaches can be used to solve such problems. In this subsection, we provide the basic framework of a divide-and-conquer algorithm based on the concepts we introduced so far.

Fix a graph class $\mathbb{G}$ for which a $\sqrt{\cdot}$-separator theorem for the counting measure and with constants α and β (see Definition 4.1.4) is known. We consider a vertex selection graph problem $\mathcal{G}$ that is glueable with the color sets C_0 and C_1. Let (G, ξ) be a precolored graph. The evaluation of the term OPT(G, ξ) (see Definition 4.1.8) can be done recursively according to the strategy given in Fig. 4.3. The correctness of this algorithm is immediate since, by definition of glueability, we know that Eq. (4.1) from Definition 4.1.10 holds for OPT(G, ξ).

We now give a precise running time analysis of this algorithm. For this analysis, the sizes of the subproblems, i.e., the sizes of the precolored graphs $(G_A^{\xi,\zeta}, \eta_A^{\xi,\zeta})$ and $(G_B^{\xi,\zeta}, \eta_B^{\xi,\zeta})$ which are used in the recursion, play a crucial role. A particularly nice situation is given for the following problems.

Definition 4.1.14 *A glueable vertex selection problem is called* slim *if, for* $X \in \{A, B\}$ *the subgraphs* $G_X^{\xi,\zeta}$ *in Definition 4.1.10 can be chosen such that they are only by a constant number of vertices larger than* $G[V_X]$, *i.e., if there exists an* $\gamma \geq 0$ *such that* $|V(G_X^{\xi,\zeta})| \leq |V_X| + \gamma$ *for all* ξ *and* ζ.

Note that the proof of Lemma 4.1.11 shows that both VERTEX COVER and INDEPENDENT SET are slim with $\gamma = 0$, whereas DOMINATING SET is not.

The following proposition gives the running time of the algorithm in Fig. 4.3 in terms of the parameters of the separator theorem used and the vertex selection problem considered. In order to assess the time required for the given divide-and-conquer algorithm, we use the following abbreviations for the running times of certain subroutines:

- $T_S(n)$ denotes the time to find a separator in an n-vertex graph from the class $\mathbb{G}$.
- $T_M(n)$ denotes the time to construct the precolored subgraphs $(G_A^{\xi,\zeta}, \eta_A^{\xi,\zeta})$ (for $X \in \{A, S, B\}$ and fixed ζ, ξ) of an n-vertex graph from the class $\mathbb{G}$.

In the following, we assume that all these functions are polynomials.

Proposition 4.1.15 *Let* $\mathbb{G}$ *be a graph class for which a* $\sqrt{\cdot}$*-separator theorem for the counting measure with constants* α *and* β *is known and let* $\mathcal{G}$ *be a vertex selection problem with* λ *colors.*

Then, for a precolored graph (G, ξ) *with* $G \in \mathbb{G}$, $\mathrm{OPT}_{\mathcal{G}}(G, \xi)$ *can be computed in time*

$$O(\, d_\varepsilon \cdot 2^{\theta(\alpha',\beta,\lambda)\sqrt{n}} \cdot q(n)\,), \quad \textit{where} \quad \theta(\alpha', \beta, \lambda) = \log(\lambda) \cdot \frac{\beta}{1 - \sqrt{\alpha'}}.$$

Here, $\alpha' = \alpha + \varepsilon$ *for any* $\varepsilon \in (0, 1 - \alpha)$, d_ε *is some constant depending on* ε, *and* $q(\cdot)$ *is some polynomial.*

If, however, $\mathcal{G}$ *is slim then the running time for the computation is*

$$O(\, 2^{\theta(\alpha,\beta,\lambda)\sqrt{n}} q(n)\,).$$

Proof: Let $T(n)$ denote the running time to compute $\mathrm{OPT}_{\mathcal{G}}(G, \xi)$ for a precolored graph (G, ξ) with n vertices. In general, the recurrence we have to solve in order to compute an upper bound on $T(n)$ for the divide-and-conquer algorithm in Fig. 4.3 then reads

$$T(n) \leq \lambda^{\beta\sqrt{n}} \cdot (T_M(n) + 2\,T(\alpha n + \beta\sqrt{n}) + T(\beta\sqrt{n})) + T_S(n),$$

which can be seen as follows. By assumption, a separation (V_A, V_S, V_B) of G can be found in time $T_S(n)$. Since the size of the separator is upperbounded by $\beta\sqrt{n}$ and there are λ colors, there are $\lambda^{\beta\sqrt{n}}$ many passes through a loop which checks all possible assignments to separator vertices. In each pass through the loop, three instances of smaller subproblems have to be constructed and evaluated; the subinstances of A and B having size bounded by $\alpha n + \beta\sqrt{n}$ and the subinstance of S having size bounded by $\beta\sqrt{n}$. This yields the term $T_M(n) + 2 \cdot T(\alpha n + \beta\sqrt{n}) + T(\beta\sqrt{n})$. Note that the functions $T_M(n)$ and $T_S(n)$ are polynomials by our general assumption.

For every $\varepsilon \in (0, 1-\alpha)$, letting $n_\varepsilon := (\frac{\beta}{\varepsilon})^2$, we have

$$(\alpha+\varepsilon)n \geq \alpha n + \beta\sqrt{n} \tag{4.2}$$

for $n \geq n_\varepsilon$. Hence, by setting $\alpha' = \alpha + \varepsilon$, we can simplify the above recurrence to

$$T(n) \leq \lambda^{\beta\sqrt{n}}(3 \cdot T(\alpha' n) + T_M(n)) + T_S(n)$$

for some $\alpha' < 1$. The recursion depth is $n' = \log_{1/\alpha'}(n)$, and we get the upper bound

$$\begin{aligned} T(n) &\leq \lambda^{\beta\sum_{i=0}^{n'}\sqrt{\alpha'^i n}} \cdot \ 3^{n'}\, T(1)\, T_M(n) + n' \cdot T_S(n) \\ &\leq \lambda^{\beta/(1-\sqrt{\alpha'})\cdot\sqrt{n}} \cdot \ q(n), \end{aligned}$$

which holds for all $n \geq n_\varepsilon$ for the polynomial

$$q(n) := 3^{\log_{1/\alpha'} n} T(1)\, T_M(n) + \log_{1/\alpha'}(n) \cdot T_S(n). \tag{4.3}$$

The claim follows with $d_\varepsilon := T(n_\varepsilon)$.

Now, consider the case where $\mathcal{G}$ is slim. In this situation the recursive subproblems all have size bounded by $\alpha n + \gamma$. Hence, we have

$$T(n) \leq \lambda^{\beta\sqrt{n}} \cdot (3T(\alpha n + \gamma) + T_M(n)) + T_S(n).$$

The size of the problem after r recursion steps obviously is:

$$\alpha^r n + \alpha^{r-1}\gamma + \cdots + \alpha\gamma + \gamma \leq \alpha^r n + \frac{1}{1-\alpha}\gamma.$$

Since the recursion will stop after $n' = \log_{1/\alpha} n$ recursive calls (leaving us with subproblems of at most constant size), we can further estimate:

$$\begin{aligned} T(n) &\leq \lambda^{\beta\sum_{i=0}^{n'}\sqrt{\alpha^i n + \gamma/(1-\alpha)}}\, 3^{n'}\, T(1)\, T_M(n) + n' \cdot T_S(n) \\ &\leq \lambda^{\beta\cdot(\sum_{i=0}^{n'}\sqrt{\alpha^i n} + (n'+1)\cdot\sqrt{\gamma/(1-\alpha)})}\, 3^{n'}\, T(1)\, T_M(n) + n' \cdot T_S(n) \\ &\leq \lambda^{\beta/(1-\sqrt{\alpha})\cdot\sqrt{n}} q(n) \end{aligned}$$

for some polynomial $q(\cdot)$. More precisely, we can estimate

$$q(n) \leq \lambda^{\beta\sqrt{\gamma/(1-\alpha)}(n'+1)} 3^{n'} T(1)\, T_M(n) + \log_{1/\alpha'}(n) \cdot T_S(n)\,.$$

□

4.2 Planar Graphs: $2^{O(\sqrt{k})}$-Algorithms Based on Separation

We now investigate the design of fixed-parameter algorithms using the considerations we have made up to this point: If the considered parameterized problem has a problem kernel of size dk, we will see that we can easily obtain fixed-parameter algorithms whose exponential part of the running time is of the form $2^{O(\sqrt{k})}$. In particular, this means that we obtain fixed-parameter algorithms for all glueable vertex selection problems that admit a linear problem kernel on planar graphs. Examples for such problems are given by par-VERTEX COVER, par-INDEPENDENT SET, or par-DOMINATING SET on planar graphs. We follow parts of [9].

4.2.1 How (Linear) Problem Kernels Help

Combining problem kernel reduction (see Chapter 2) with the divide-and-conquer approach for glueable vertex selection problems (see Subsection 4.1.2), we obtain the following result.

Theorem 4.2.1 *Assume the following:*

- *Let* $\mathbb{G}$ *be a graph class for which a* $\sqrt{\cdot}$*-separator theorem for the counting measure with constants* α *and* β *is known,*
- *let* $\mathcal{G}$ *be a vertex selection problem which is glueable with* λ *colors, and*
- *suppose that* par-$\mathcal{G}$ *admits a problem kernel of size* $p(k)$ *on* $\mathbb{G}$ *computable in time* $q_{kernel}(n,k)$.

Then, there is an algorithm to solve par-$\mathcal{G}$ *in time*

$$O\big(\, d_\varepsilon \cdot 2^{\theta(\alpha',\beta,\lambda)\sqrt{p(k)}} \cdot q(k) + q_{kernel}(n,k)\,\big), \quad \textit{where} \quad \theta(\alpha',\beta,\lambda) = \log(\lambda)\cdot\frac{\beta}{1-\sqrt{\alpha'}}. \tag{4.4}$$

Here, $\alpha' = \alpha+\varepsilon$ *for any* $\varepsilon \in (0, 1-\alpha)$, d_ε *is a constant depending on* ε, *and* $q(\cdot)$ *is a polynomial.*

If, however, $\mathcal{G}$ *is slim then* par-$\mathcal{G}$ *can be solved in time* $O\big(\, 2^{\theta(\alpha,\beta,\lambda)\sqrt{p(k)}} q(k) + q_{kernel}(n,k)\,\big)$.

Proof: The claim directly follows from Proposition 4.1.15 applied to the problem kernel. □

In particular, Theorem 4.2.1 shows that for glueable and slim vertex selection problems that admit a *linear* problem kernel of size dk on planar graphs, we get an algorithm of running time

$$O\big(\, 2^{\theta(\alpha,\beta,\lambda,d)\sqrt{k}} q(k) + q_{\mathrm{kernel}}(n,k)\,\big), \quad \text{where} \quad \theta(\alpha,\beta,\lambda,d) = \log(\lambda)\cdot\frac{\beta}{1-\sqrt{\alpha}}\cdot\sqrt{d}. \tag{4.5}$$

Obviously, the choice of the separator theorem has a decisive impact on the constants of the corresponding algorithms. In particular, our running time analysis shows that the ratio $r(\alpha,\beta) := \beta/(1-\sqrt{\alpha})$ has a direct and significant influence on the running time. In Table 4.1, this ratio is computed for the various $\sqrt{\cdot}$-separator theorems. The best ratio obtained there is due to the theorem of Djidjev and Venkatesan [77] with $\alpha = 2/3$ and $\beta = \sqrt{2/3} + \sqrt{4/3}$. Here, we get $r(\alpha,\beta) \approx 10.74$. This ratio is used explicitly in the following corollary.[2]

Corollary 4.2.2 *Letting* $q(k) = k\cdot\log(k)$, *we obtain that:*

(i) par-VERTEX COVER *on planar graphs can be solved in time* $O(2^{15.19\sqrt{k}}q(k) + kn)$.

(ii) par-INDEPENDENT SET *on planar graphs can be solved in time* $O(2^{21.48\sqrt{k}}q(k) + n^2)$.[3]

(iii) par-DOMINATING SET *on planar graphs can be solved in time* $O(2^{340.79\sqrt{k}}q(k) + n^3)$.

[2] As usual, in the following corollary, n will always denote the number of vertices of an input graph and k will refer to the corresponding problem parameter according to Definition 1.3.2.

[3] Using the problem kernelization in Remark 2.1.5, we even get a running time of the form $O(2^{O(\sqrt{k})} + n)$ (with a slightly worse constant in the exponent).

Proof: The running times are obtained by plugging in the various problems specific values in Eq. (4.5): We use the problem kernels, where $d = 2$ for par-VERTEX COVER (see Corollary 2.1.8), $d = 4$ for par-INDEPENDENT SET (see Proposition 2.1.4), and $d = 335$ for par-DOMINATING SET. Besides, we take the constants $\lambda = 2$ for VERTEX COVER and INDEPENDENT SET (see Lemma 4.1.11), and $\lambda = 3$ for DOMINATING SET (see Remark 4.1.12).

To obtain the claimed running times for par-VERTEX COVER and par-INDEPENDENT SET we can directly plug in the corresponding values in Eq. (4.5). Since DOMINATING SET is not slim, we have to use a value $r(\alpha', \beta)$ for some $\alpha' = \alpha + \varepsilon$. Hence, for a suitable $\varepsilon > 0$, we might use $r(\alpha', \beta) \approx 10.75$. The polynomial term $q(k)$ can be obtained from Equation (4.3). It is easy to see that in our setting we get $q(k) = k \cdot \log(k)$. □

As explained in the following remark, Theorem 4.2.1 is not only interesting in the case of planar graph problems.

Remark 4.2.3 Theorem 4.2.1 yields a time $O(2^{O(\sqrt{gk})} + k^3 + kn)$ algorithm for par-VERTEX COVER on the class $\mathbb{G}(S_g)$ of graphs of genus bounded by g (see Section 1.2). This is true since par-VERTEX COVER admits a linear problem kernel (on $\mathbb{G}_g$) and since the existence of a separator of size $O(\sqrt{gn})$ for n-vertex graphs from $\mathbb{G}(S_g)$ was proven by Djidiev [76].

We finish this subsection by a brief discussion of the practical and theoretical value of the "$2^{O(\sqrt{k})}$-algorithms" obtained in Theorem 4.2.1. Here, "$2^{O(\sqrt{k})}$-algorithm" shall refer to sublinear-exponential fixed-parameter algorithm with running time of the form $2^{O(\sqrt{k})} n^{O(1)}$.

On the negative side, the bases of the exponential terms in the running time obtained there (see the concrete running times in Corollary 4.2.2) are admittedly (very) bad and, hence, will most likely be too large for practical purposes. We mention in passing that some attempts to (somewhat) improve the bases of the running times can be found in [9, Technical Report] or [94]. However, even after these slight improvements these $2^{O(\sqrt{k})}$-algorithms probably remain uninteresting from a practical point of view. We will reconsider the design of $2^{O(\sqrt{k})}$-algorithms for planar graph problems in Chapter 5. There, we will derive such algorihms with *considerably* smaller bases of the exponential term in the running time using the concept of tree decompositions. This concept is more powerful and seems to be much more promising from a practical point of view.

On the positive side, the results in Theorem 4.2.1 and Corollary 4.2.2 are of theoretical importance for two reasons:

- Fristly, these are—to the best of our knowledge—the first fixed-parameter algorithms which have a sublinear term in the exponent of the exponential term of the running time, i.e., where the running time is of the form $2^{o(k)} n^{O(1)}$.
- Secondly, we will see in the next subsection, that the algorithms given in Corollary 4.2.2 are asymptotically optimal. There we will prove that an algorithm of running time $2^{o(\sqrt{k})} n^{O(1)}$ is not possible for any of the problems given in Corollary 4.2.2, unless 3 SAT $\in$ DTIME$(2^{o(n)})$, n being the number of variables of a 3 SAT formula (see Theorem 4.2.10 below). In classical complexity theory, this fact, i.e., that 3 SAT can be solved in exponential time with a sublinear exponent in n, is considered to be unlikely.

4.2.2 Lower Bounds

Recall from the Introduction (see Section 1.1) that the questions whether SAT or 3 SAT can be solved in running time $2^{o(n)}$ are fundamental in the theory of parameterized complexity, since they are closely related to the question whether FPT = W[1]. More precisely, FPT = W[1] implies that 3 SAT $\in$ DTIME($2^{o(n)}$), n being the number of variables [1]. Conversely, it is stated as the main result in [53] that SAT $\in$ DTIME($2^{o(n)}$), n being the number of variables, implies FPT = W[1].

We now provide a lower bound for the running time of a fixed-parameter algorithm for par-VERTEX COVER. Our working hypothesis for these lower bounds again will be the assumption that 3 SAT $\notin$ DTIME($2^{o(n)}$), where n is the number of variables. This will then lead to similar lower bounds for fixed-parameter algorithms for par-VERTEX COVER, par-INDEPENDENT SET, and par-DOMINATING SET on planar graphs. Our first result relies on a close investigation of the classical reduction of 3 SAT to VERTEX COVER on graphs of degree bounded by three (abbreviated by VERTEX COVER-3) that is due to Garey, Johnson, and Stockmeyer [101, Theorem 2.6]. The following result was first observed by Cai and Juedes [53, Lemma 4.2].[4]

Proposition 4.2.4 *If* par-VERTEX COVER-3 *can be solved in time* $2^{o(k)}n^{O(1)}$ *then* 3 SAT $\in$ DTIME($2^{o(m)}$)*, where* m *is the number of clauses in a* 3 SAT *formula.*

Proof: Assume that there is a time $2^{o(k)}n^{O(1)}$ algorithm for par-VERTEX COVER-3. Since $k \leq n$ we, thus, have a time $2^{o(n)}$ algorithm for VERTEX COVER-3.

Suppose we are given a 3 SAT formula ϕ with m clauses. In the classical reduction of 3 SAT to VERTEX COVER [101, Theorem 2.6], it is shown how ϕ can be transformed in time polynomial in m to a graph G of degree at most three with $n = 9m$ vertices in such a way that ϕ has a satisfying assignment if and only if G has a vertex cover of size $5m$. Hence, in order to solve 3 SAT on the instance ϕ, we only have to find a minimum vertex cover of G. Assuming the time $2^{o(n)}$ algorithm for VERTEX COVER-3, this can be done in time $2^{o(9m)}$. This shows that 3 SAT $\in$ DTIME($2^{o(m)}$). □

In order to get a similar lower bound based on the assumption that 3 SAT $\notin$ DTIME($2^{o(n)}$), where n is the number of *variables* in a 3 SAT formula, we use a "Sparsification Lemma" due to Imagliazzo, Paturi, and Zane [119]. This lemma states that for every $\varepsilon > 0$, there exists a constant d_ε such that any 3 SAT formula ϕ (with n variables) can be expressed as $\phi = \bigvee_{i=1}^{t} \phi_i$ with $t \leq 2^{\varepsilon n}$, where each ϕ_i is a 3 SAT formula with at most $d_\varepsilon n$ clauses. Using this Sparsification Lemma it is not hard to show the following statement (see [53, Lemma 4.3] for a proof).

Lemma 4.2.5 *If* 3 SAT $\in$ DTIME($2^{o(m)}$)*,* m *being the number clauses in a formula, then* 3 SAT $\in$ DTIME($2^{o(n)}$)*, where* n *is the number of variables in a formula.* □

Combining Lemma 4.2.5 with Proposition 4.2.4 we get the following.

Proposition 4.2.6 *If* par-VERTEX COVER-3 *can be solved in time* $2^{o(k)}n^{O(1)}$ *then* 3 SAT $\in$ DTIME($2^{o(n)}$)*, where (in the latter case)* n *is the number of variables in a* 3 SAT *formula.* □

[4] As usual, in the following results for a parameterized graph problem, n will always denote the number of vertices of an input graph and k will refer to the corresponding problem parameter according to Definition 1.3.2.

Remark 4.2.7 In fact, Cai and Juedes showed that the converse is also true, i.e., if 3 SAT $\in$ DTIME$(2^{o(n)})$, n being the number of variables, then there is a time $2^{o(k)}n^{O(1)}$ algorithm for par-VERTEX COVER-3 [53, Theorem 4.3]. Moreover, a similar statement can be easily generalized to all problems that are MAXSNP-complete (under so-called L-reduction as introduced in [162]). More precisely, Cai and Juedes [53, Theorem 4.4] show that, for a MAXSNP-complete problem $\mathcal{Q}$, the induced parameterized problem par-$\mathcal{Q}$ can be solved in time $2^{o(k)}n^{O(1)}$ with witness if and only if 3 SAT $\in$ DTIME$(2^{o(n)})$, where n is the number of variables. Examples for MAXSNP-hard problems are given by VERTEX COVER, INDEPENDENT SET, DOMINATING SET (on graphs of degree bounded by 3), or MAX C-SAT.

To obtain the corresponding lower bounds for our *planar* graph problems, we need the following notion of a parameterized reduction.

Definition 4.2.8 *Let* $\mathcal{L}_1 \subseteq \Sigma_1^* \times \mathbb{N}$ *and* $\mathcal{L}_2 \subseteq \Sigma_2^* \times \mathbb{N}$ *be parameterized problems. We say that* $\mathcal{L}_1$ *is* parameterized reducible *to* $\mathcal{L}_2$ *if there exist a computable function* $\Psi : \mathbb{N} \to \mathbb{N}, k \mapsto k'$ *and a function* $\Phi : \Sigma_1^* \times \mathbb{N} \to \Sigma_2^*, (x, k) \mapsto x'$ *such that*

(i) Φ *is computable in time* $f(k) \cdot |x|^{O(1)}$ *for some function* f*, and*

(ii) $(x, k) \in \mathcal{L}_1$ *if and only if* $(x', k') \in \mathcal{L}_2$.

If such a parameterized reduction exists, we write $\mathcal{L}_1 \leq_\Psi \mathcal{L}_2$.

In the original (classical) reductions from VERTEX COVER to VERTEX COVER on planar graphs, to INDEPENDENT SET on planar graphs, or to DOMINATING SET on planar graphs, Garey, Johnson, and Stockmeyer [101, Theorem 2.7] use the idea of so-called "crossover gadgets." By a careful investigation on how the problem parameter k transforms in this classical reduction, it is not hard to show the following result which was first observed by Cai and Juedes [53]. We refer to [53, Lemma 5.1] for an easy proof.

Lemma 4.2.9 *There exist parameterized reductions*

(i) par-VERTEX COVER-3 $\leq_{k \mapsto k^2}$ (par-VERTEX COVER *on planar graphs*),

(ii) par-VERTEX COVER-3 $\leq_{k \mapsto k^2}$ (par-INDEPENDENT SET *on planar graphs*), *and*

(iii) par-VERTEX COVER-3 $\leq_{k \mapsto k^2}$ (par-DOMINATING SET *on planar graphs*).

All reductions can be carried out in polynomial time.

These parameterized reductions in combination with Proposition 4.2.6 yield the mentioned lower bounds for our threesome of planar graph problems (see [53, Lemma 5.1]).

Theorem 4.2.10 *If* par-VERTEX COVER, par-INDEPENDENT SET, *or* par-DOMINATING SET *is solvable in time* $2^{o(\sqrt{k})}n^{O(1)}$ *on planar graphs, then* 3 SAT $\in$ DTIME$(2^{o(n)})$*, where (in the latter case)* n *is the number of variables in a* 3 SAT *formula.*

Proof: Using the parameterized reduction given in Lemma 4.2.9, it is easy to see that an algorithm of running time $2^{o(\sqrt{k})} n^{O(1)}$ can be used to solve the par-VERTEX COVER-3 problem in time $2^{o(\sqrt{k^2})} n^{O(1)}$. This implies, by Proposition 4.2.6, that 3 SAT $\in$ DTIME$(2^{o(n)})$. □

4.3 Beyond Planar Graphs: Algorithms on Disk Graphs

We now turn our attention to so-called disk graphs. A disk graph is an intersection graph (i.e., a graph in which the vertices correspond to geometric objects and there is an edge between two objects if the objects intersect) in the two-dimensional space, where the intersecting objects are disks. A famous result by Koebe [130] shows that planar graphs are equivalent to the class of coin graphs, i.e., disk graphs where disks are not allowed to overlap, but only to touch. In this sense, disk graphs are a generalization of planar graphs. Note that not all graphs can be represented by disks. A brief introduction to disk graphs is given in Subsection 4.3.1. For the case of disk graphs, as in the previous section, we will pursue the strategy of combining a geometric version of reduction to problem kernel (see Subsection 4.3.2) with a divide-and-conquer approach based on an appropriate separator theorem.

Unfortunately, for disk graphs, so far such separator theorems are known only for intersection graphs of so-called "τ-ply neighborhood systems" [88, 147, 148] (see Subsection 4.3.3.1). With respect to general disk graphs, we quote from the introduction of Hunt *et al.* [118]:

> *"The [...] drawback is that problems such as maximum independent set and minimum dominating set [...]* **cannot** *be solved at all by the separator approach. This is because an arbitrary (unit) disk graph of n nodes can have a clique of size n."*

The key result in this section is to show a way out of this dilemma by proving a new type of "geometric separator theorem" which holds for disk graphs with bounded radius ratio (see Subsection 4.3.3). Our geometric separator theorem can be seen as a generalization of (classical) separator theorems, where the guarantee is not on the size of the separator in terms of its number of vertices but in terms of the space occupied by its disks.

In Subsection 4.3.4 the results are used to optimally solve par-INDEPENDENT SET on disk graphs of bounded radius ratio. The problem is motivated by numerous applications ranging from map labeling problems [2] to the area of frequency assignment problems in cellular networks [142]. In the latter case, one considers a set of antennas which transmit data on a given frequency to their local environment. Assuming that this environment can be modeled by a disk centered at the position of the antenna, the task to determine the maximum number of antennas which can operate simultaneously without any conflicting interferences using the same frequency becomes a maximum INDEPENDENT SET problem on a disk graph.[5] We provide an algorithm running in time $2^{O(\sqrt{k}\log(n))}$, which is—to our knowledge—the first algorithm with running time bounded by a function with a the exponent sublinear in k. Clearly, the algorithm's running time

[5]Note that we always consider disk graphs with given representation, i.e., with given set of disks in the plane. This makes sense, since most applications which are modeled by disk graphs already provide this representation in a very natural way.

must be considered to be impractical, but it is relevant from a theoretical point of view, since in the worst-case (i.e., when k equals n) we obtain a running time which is considered to be very unlikely on general graphs. More precisely, we obtain an algorithm of running time $2^{e(n)}$ with the sublinear term $e(n) = \sqrt{n}\log(n)$; this cannot be achieved for general graphs unless 3 SAT $\in$ DTIME$(2^{o(n)})$, where n is the number of variables in a 3 SAT formula [119].

Moreover, in the special case of disk graphs with ϑ-precision (i.e., the centers of all disks are at mutual distance at least $\vartheta > 0$)[6], we can even show that par-INDEPENDENT SET is in FPT. As in the planar case, we obtain a running time of $O(2^{O(\sqrt{k})} + n)$. A similar result holds for par-VERTEX COVER, and par-DOMINATING SET on disk graphs with ϑ-precision. In this section, we partly follow [10].

4.3.1 Disk Graphs, Lebesgue Graph Measure and Covering Grids

(Disk) Intersection Graphs. Our subject to explore are intersection graphs of geometric objects in the plane, namely of disks. We assume that each point z of the plane is determined by its x any y coordinates and the plane is equipped by the standard distance $d(z, z') = \sqrt{(x - x')^2 + (y - y')^2}$. The plane distance of two geometric objects S and S' is determined as the infimum distance among all pairs of $z \in S$ and $z' \in S'$. If $\mathcal{S} = \{S_1, \ldots, S_n\}$, $S_i \subseteq \mathbb{R}^2$ is a collection of geometric objects, we denote by $\bigcup \mathcal{S} = \bigcup_{i=1}^{n} S_i$ the union of $\mathcal{S}$.

Definition 4.3.1 *For a collection $\mathcal{S}$ of $n$geometric objects, let $G_{\mathcal{S}} = (V_{\mathcal{S}}, E_{\mathcal{S}})$ denote the* intersection graph *of $\mathcal{S}$, i.e., $V_{\mathcal{S}} = \{v_1, \ldots, v_n\}$ and $E_{\mathcal{S}} = \{(v_i, v_j) \mid S_i \cap S_j \neq \emptyset\}$. The collection $\mathcal{S}$ is called the* representation *of $G_{\mathcal{S}}$. Moreover, for a subset $\mathcal{S}' \subseteq \mathcal{S}$, we denote by $V_{\mathcal{S}'} \subseteq V_{\mathcal{S}}$ the subset of vertices induced by $\mathcal{S}'$, i.e., $V_{\mathcal{S}'} = \{v_i \mid S_i \in \mathcal{S}'\}$. Finally, $G_{\mathcal{S}'} := G_{\mathcal{S}}[\mathcal{S}'] := G_{\mathcal{S}}[V_{\mathcal{S}'}]$ is the subgraph of $G_{\mathcal{S}}$ induced by the set of vertices $V_{\mathcal{S}'}$.*

In our setting, a disk $D \subseteq \mathbb{R}^2$ is specified by a triple $(r, x, y) \in \mathbb{R}^3$, where (x, y) are coordinates of the center of the disk in the Euclidean plane and r is its radius.

Definition 4.3.2 *The graph class of* disk graphs*, denoted by $\mathbb{DG}$ is the set of all graphs G, for which we find a collection of disks $\mathcal{D} = \{D_1, \ldots, D_n\}$ such that $G = G_{\mathcal{D}}$.*

Note that for given collection $\mathcal{D}$, the graph $G_{\mathcal{D}}$ is given with a natural embedding in the plane, where v_i sits in the position of the center of D_i.

Disk graphs are a generalization of planar graphs: A *coin graph* is a disk graph $G_{\mathcal{D}} \in \mathbb{DG}$, where the intersection of any two disks $D_1, D_2 \in \mathcal{D}$ contains at most one point in $\mathbb{R}^2$, i.e., the disks can only touch each other. It is not hard to see that every coin graph is a planar graph. A remarkable result by Koebe [130] states that the converse is also true, i.e., that every planar graph is a coin graph. This result implies that every planar graph admits a so-called "*straight-line embedding*" in the plane, i.e., an embedding in the plane where each edge is mapped to a straight line segment (for details see Fáry [90] and Tutte [185, 186]).

We want to focus on another subclass of general disk graphs.

[6] From an application point of view this is a realistic scenario, since, e.g., for the frequency assignment problem, we may assume that all antennas have a certain width.

Definition 4.3.3 *The class of disk graphs of* bounded radius ratio ρ *is the subclass* $\mathbb{DG}_\rho \subset \mathbb{DG}$ *of all graphs* $G \in \mathbb{DG}$ *which admit a representation* $\mathcal{D} = \{D_1, \ldots, D_n\}$*, such that*

$$(\max_{i=1,\ldots n} r_i)/(\min_{i=1,\ldots n} r_i) \leq \rho,$$

where r_i *denotes the radius of disk* D_i*. The parameter* ρ *is called* radius ratio.

By a rescaling argument, for a graph $G \in \mathbb{DG}_\rho$ with representation $\mathcal{D}$, we can always achieve, that the smallest disk in $\mathcal{D}$ has radius one and, hence, all radii being upper bounded by ρ. The following restriction of $\mathbb{DG}_\rho$ was introduced in [118, Definition 3.2].

Definition 4.3.4 *A collection* $\mathcal{D}$ *is said to have* ϑ-precision *(for some* $\vartheta > 0$*) if all centers of disks are pairwise at least* ϑ *apart.*[7] *By* $\mathbb{DG}_{\rho,\vartheta}$ *we denote the subclass of* $\mathbb{DG}_\rho$ *of disk graphs* $G_\mathcal{D}$ *that allow a representation* $\mathcal{D}$ *with radii in* $[1, \rho]$ *and* ϑ*-precision.*

Throughout the rest of this section, we assume that a disk graph G is given together with its representation witnessing its membership in $\mathbb{DG}$, $\mathbb{DG}_\rho$ or $\mathbb{DG}_{\rho,\vartheta}$, respectively. This make sense from an application point of view, since the graph is usually derived from the placement of real objects in the space, and it is natural that these objects have bounded size as well as distinguishable placement.

The Lebesgue Graph Measure. Recall Definition 4.1.2 of a graph measure. Disk graphs allow for a specific graph measure. We use the standard Lebesgue measure μ in $\mathbb{R}^2$ as follows: For a Lebesgue measurable set $S \subseteq \mathbb{R}^2$, $\mu(S)$ denotes the *size* of S, i.e., the space in $\mathbb{R}^2$ occupied by S. In particular, for a collection of disks $\mathcal{D} = \{D_1, \ldots, D_n\}$, let $\mu(\mathcal{D}) = \mu(\bigcup \mathcal{D})$ be the space covered by the union of the disks $D_1, \ldots, D_n$.

Definition 4.3.5 *The* Lebesgue measure $\mu(\cdot)$ *assigning to a disk graph* $G_\mathcal{D}$ *with representation* $\mathcal{D}$ *the value* $\mu(G_\mathcal{D}) = \mu(\mathcal{D})$ *is a graph measure for* $\mathbb{DG}$*, when we restrict the subgraph ordering to* $G_\mathcal{D} \subseteq G_{\mathcal{D}'} \Leftrightarrow \mathcal{D} \subseteq \mathcal{D}'$*.*

Remark 4.3.6 The value $\mu(G_\mathcal{D})$ can be computed in time polynomial in $n := |\mathcal{D}|$. In a naive approach[8], we could compute all (at most $2n^2$) intersection points of the n disks. Then, one forms a grid consisting of $O(n^4)$ rectangles, by drawing a horizontal and a vertical line through each of the intersection points. Now each of the rectangles either is completely covered by the disks, does not intersect with any disk, or is partially covered by disks (that do not intersect in this rectangle!). Hence, we can compute the space covered by the disks for each rectangle (by a simple formula) in linear time.

[7]It is clear, that, by a rescaling argument, all disk graphs have a representation with ϑ-precision. However, only some of them allow a representation with radii in $[1, \rho]$ and ϑ-precision.

[8] A sweep-line algorithm could perform much better. For the rest of the section, however, it is sufficient that we can compute $\mu(G_\mathcal{D})$ in polynomial time.

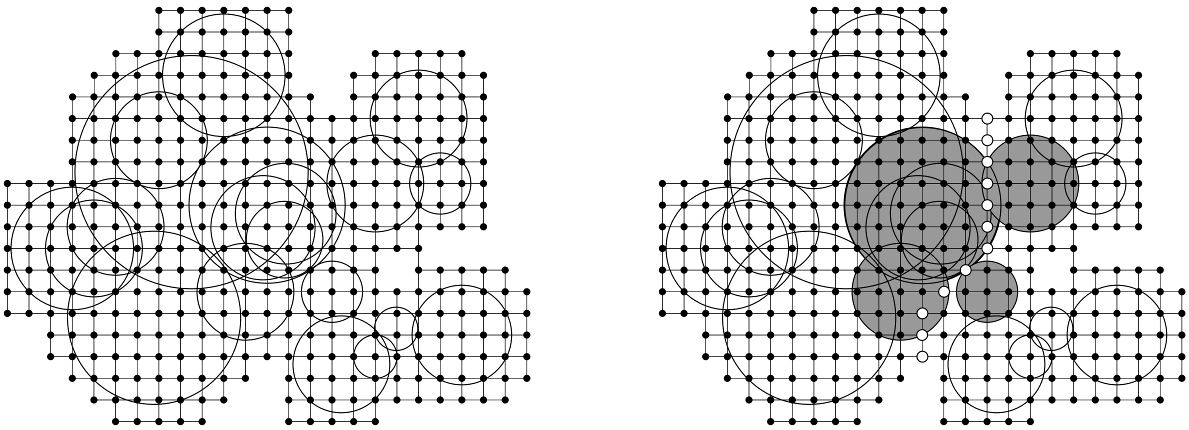

Figure 4.4: The left-hand diagram shows the construction of the covering grid $H^\delta_{\mathcal{D}}$ induced by a collection of disks $\mathcal{D}$. The right-hand diagram illustrates the construction of a geometric separator according to the algorithm in Fig. 4.5 (see Theorem 4.3.19): In a first step, a $\sqrt{\cdot}$-separator theorem on planar graphs determines a small separator W_S for the covering grid. Assume that corresponding vertices W_S are the ones that are highlighted. In this particular example, the disks that will be used for the geometric separator $\mathcal{D}_S$ are the marked ones. Observe that this geometric separator is not small in the number of disks used, but in the area covered by the disks.

Covering Grids For the proof of our geometric separator theorem it will be important to "translate" a set of disks into a planar graph. This will be established by the so-called "covering grid," which is a subgraph of an infinite grid.

Fix an arbitrary constant $\delta > 0$, and consider the infinite grid of span δ as the planar graph $H^\delta = (W^\delta, E^\delta)$ with vertices $W^\delta = \{w_{i,j} \mid i,j \in \mathbb{Z}\}$ and edges $E^\delta = \{(w_{i,j}, w_{k,l}) \mid |i-j|+|k-l| = 1\}$. The canonical (straight-line) embedding of H^δ is given by putting vertex $w_{i,j}$ at the coordinates $(i\delta, j\delta)$. In the following, for convenience, we will not distinguish between a vertex $w \in W^\delta$ and the corresponding point in the plane. The set of faces $\mathcal{F}^\delta$ of H^δ contains all closed squares $F^\delta_{i,j} = [i\delta, (i+1)\delta] \times [j\delta, (j+1)\delta] \subseteq \mathbb{R}^2$. For a grid vertex $w \in W^\delta$, we define the *face neighborhood* $\hat{N}(w) := \{F \in \mathcal{F}^\delta \mid w \in F\}$. Similarly, for $W' \subseteq W^\delta$, let $\hat{N}(W') = \bigcup_{w \in W'} \hat{N}(w)$.

Definition 4.3.7 *For a collection of disks $\mathcal{D} = \{D_1, \dots, D_n\}$, we define $H^\delta_{\mathcal{D}}$ to be the smallest subgraph of the infinite grid H^δ induced by a set of grid points which completely covers all disks in $\mathcal{D}$. We call $H^\delta_{\mathcal{D}}$ the* covering grid *(of span δ) for $\mathcal{D}$.*

In other words, if we define the set of faces hit by $\mathcal{D}$ as $\mathcal{F}^\delta_{\mathcal{D}} = \{F \in \mathcal{F}^\delta \mid F \cap \bigcup \mathcal{D} \neq \emptyset\}$, and if $W^\delta_{\mathcal{D}}$ is the set of all grid points of $\mathcal{F}^\delta_{\mathcal{D}}$, then the covering grid $H^\delta_{\mathcal{D}}$ is the subgraph of H^δ induced by $W^\delta_{\mathcal{D}}$.

An example which illustrates the construction of the covering grid $H^\delta_{\mathcal{D}}$ is given in the left-hand diagram in Fig. 4.4.

4.3.2 Geometric Problem Kernelizations

We already investigated par-VERTEX COVER, par-INDEPENDENT SET, and par-DOMINATING SET on planar graphs and observed that all problems admit a linear problem kernel (see Corollary 2.1.8, Proposition 2.1.4 and Theorem 2.2.1). For par-VERTEX COVER, the planarity assumption was indispensable. In contrast, for par-INDEPENDENT SET and par-DOMINATING SET, the existence of a linear kernel relied on this assumption.

On disk graphs, we can prove a geometric version of a problem kernel for these problems. By this, we mean that the size of the reduced instance is upper bounded by $O(k)$, when measured by the (Lebesgue) measure $\mu(\cdot)$ instead of the counting measure $|\cdot|$ (see Example 4.1.3).

Proposition 4.3.8 *For* par-INDEPENDENT SET *on* $\mathbb{DG}_\rho$ *there exists a "geometric" problem kernel, i.e., there is a procedure that in polynomial time transforms an instance* $(G_{\mathcal{D}}, k)$ *to an instance* $(G_{\mathcal{D}'}, k)$*, such that* $(G_{\mathcal{D}}, k) \in$ par-INDEPENDENT SET *if and only if* $(G_{\mathcal{D}'}, k) \in$ par-INDEPENDENT SET *and*

$$\pi k \;\leq\; \mu(G_{\mathcal{D}'}) \;\leq\; 9\pi\rho^2 k.$$

Proof: Let be given an instance $(G_{\mathcal{D}}, k)$, together with the representation $\mathcal{D}$. Recall, that due to our assumptions all disks have radius in the range $[1, \rho]$ and $G_{\mathcal{D}}$ is naturally embedded in the plane with respect to $\mathcal{D}$.

Observe first, that $\mu(G_{\mathcal{D}}) > 9\pi\rho^2 k$ implies that $(G_{\mathcal{D}}, k) \in$ par-INDEPENDENT SET. We use the fact that $\mu(\mathcal{D}[N(v)]) \leq (3\rho)^2\pi$ for any vertex $v \in V$, i.e., the neighborhood of any vertex may occupy the space at most $9\pi\rho^2$.

And, secondly, if $\mu(G_{\mathcal{D}}) < \pi k$, then $(G_{\mathcal{D}}, k) \notin$ par-INDEPENDENT SET, since the representation of any independent set of k vertices needs space at least πk.

Hence,

$$\Phi((G_{\mathcal{D}}, k)) := \begin{cases} (G_{\mathcal{D}'}, k) & \text{if } \mu(G_{\mathcal{D}}) > 9\pi\rho^2 k \\ (G_{\emptyset}, k) & \text{if } \mu(G_{\mathcal{D}}) < \pi k \\ (G_{\mathcal{D}}, k) & \text{otherwise,} \end{cases} \tag{4.6}$$

where $\mathcal{D}'$ is the set of k disjoint disks of radius one, yields the desired transformation. The time needed to compute the transformation Φ is determined by the time needed to compute $\mu(G_{\mathcal{D}})$, which is polynomial according to Remark 4.3.6. □

Proposition 4.3.9 *For* par-DOMINATING SET *on* $\mathbb{DG}_\rho$ *there is a procedure that in polynomial time transforms an instance* $(G_{\mathcal{D}}, k)$ *to an instance* $(G_{\mathcal{D}'}, k)$*, such that* $(G_{\mathcal{D}}, k) \in$ par-DOMINATING SET *if and only if* $(G_{\mathcal{D}'}, k) \in$ par-DOMINATING SET *and*

$$\pi k \;\leq\; \mu(G_{\mathcal{D}'}) \;\leq\; 9\pi\rho^2 k.$$

Proof: Let $(G_{\mathcal{D}}, k)$ be given together with the representation $\mathcal{D}$. If $\mu(G_{\mathcal{D}}) > 9\pi\rho^2 k$ then $(G_{\mathcal{D}}, k) \notin$ par-DOMINATING SET, since k disks can cover at most space $9\pi\rho^2 k$. Conversely, if $\mu(G_{\mathcal{D}}) < \pi k$, then $(G_{\mathcal{D}}, k) \in$ par-DOMINATING SET: The simple procedure to greedily find a dominating set of size at most k chooses, in each step, an arbitrary disk D to belong to the dominating set and removes D together with its neighbors. Since, in each step, we reduce the space occupied by the current graph by at least π, the procedure stops in at most k steps.

The claimed transformation uses a case distinction similar to (4.6). □

Again, note that neither the result in Proposition 4.3.8 nor in Proposition 4.3.9 is a problem kernel according to Definition 2.1.1, since the size of $G_{\mathcal{D}}$ is measured by the (Lebesgue) measure $\mu(\cdot)$, which, in general, is not related to the (instance) size of G. However, for disk graphs with ϑ-precision we can upper bound the counting measure by the Lebesgue measure.

Lemma 4.3.10 *Let the collection of disks $\mathcal{D}$ have ϑ-precision. Then, it holds*

$$|\mathcal{D}| \leq \frac{4}{\pi\vartheta^2}\,\mu(\mathcal{D}).$$

Proof: Observe that, for given set of n disks $\mathcal{D}$ with radius at least one and centers of mutual distance at least ϑ, the smallest value $\mu(\mathcal{D})$ is obtained by optimally placing all centers in the interior of a disk. However, such a disk must have radius at least $\frac{\vartheta\sqrt{n}}{2}$. Hence, since every disk has radius at least one, we have

$$\mu(\mathcal{D}) \geq \left(\frac{\vartheta\sqrt{n}}{2}+1\right)^2\pi \geq \frac{\pi}{4}\vartheta^2 n.$$

□

Combining this result with Propositions 4.3.8 and 4.3.9 we get the following corollaries.

Corollary 4.3.11 par-INDEPENDENT SET *on disk graphs $\mathbb{DG}_{\rho,\vartheta}$ (with ϑ-precision) admits a problem kernel of size $36(\frac{\rho}{\vartheta})^2k$, hence, it is in* FPT.

Corollary 4.3.12 par-DOMINATING SET *on disk graphs $\mathbb{DG}_{\rho,\vartheta}$ (with ϑ-precision) admits a problem kernel of size $36(\frac{\rho}{\vartheta})^2k$, hence, it is in* FPT.

Remark 4.3.13 The problem kernel reduction needs polynomial time. We can bring this down to *linear* time (at the expanse of a slightly worse problem kernel size) if the computation of $\mu(G_{\mathcal{D}})$ is done in an approximate way: Recall the notion from Definition 4.3.7. For some constant $\delta > 0$, let $H^{\delta}_{\mathcal{D}}$ be the covering grid (of span δ) for $\mathcal{D}$ and let $\mathcal{F}^{\delta}_{\mathcal{D}}$ be its faces, i.e., the set of faces of the infinite grid H^{δ} hit by $\mathcal{D}$. Observe that the set $\mathcal{F}' := \{F \in \mathcal{F}^{\delta}_{\mathcal{D}} \mid F \subseteq D\}$ can be constructed in time $O(|\mathcal{D}|)$ since, for each disk $D \in \mathcal{D}$, we need to check at most $O((\pi\rho^2)/\delta^2)$ many faces of the covering grid. Using $|\mathcal{F}'|\delta^2 \leq \mu(G_{\mathcal{D}})$ and the arguments given in the proofs of Propositions 4.3.8 and 4.3.9, we know that $(G_{\mathcal{D}}, k) \in$ par-INDEPENDENT SET and $(G_{\mathcal{D}}, k) \notin$ par-DOMINATING SET if $|\mathcal{F}'|\delta^2 > 9\pi\rho^2k$, which means that we only have to deal with the case that $|\mathcal{F}'|\delta^2 \leq 9\pi\rho^2k$. An easy computation shows that for a disk of radius at least one, the ratio between fully and partially covered grid squares of span δ is at most $c_\delta := (1+\sqrt{2}\delta)^2/(1-\sqrt{2}\delta)^2$. Hence, in the remaining case where $|\mathcal{F}'|\delta^2 \leq 9\pi\rho^2k$ we get

$$\mu(G_{\mathcal{D}}) \leq |\mathcal{F}^{\delta}_{\mathcal{D}}|\delta^2 \leq c_\delta|\mathcal{F}'|\delta^2 \leq c_\delta 9\pi\rho^2k.$$

(Observe that $c_\delta \searrow 1$ as $\delta \to 0$.)

4.3.3 A New Geometric $\sqrt{\cdot}$-Separator Theorem

In the following, we prove the key result—a new geometric $\sqrt{\cdot}$-separator theorem for general disk graphs of bounded radius ratio—that makes a divide-and-conquer strategy work.

4.3.3.1 $\sqrt{\cdot}$-Separator Theorem on Disk Graphs with ϑ-Precision

For geometric graphs, a $\sqrt{\cdot}$-separator theorem for the counting measure $|\cdot|$ (see Example 4.1.3 and Definition 4.1.4) was—to the best of our knowledge—so far only proven on the class of intersection graphs of so-called τ*-ply neighborhood systems* as studied in a series of papers by Miller, Teng, Thurston and Vavasis (see [148] for an overview).

Definition 4.3.14 *A* (2-dimensional) τ-ply neighborhood system *is a collection $\mathcal{D}$ of disks such that the intersection of any $(\tau+1)$ distinct disks in $\mathcal{D}$ is empty. By $\mathbb{DG}_{\tau\text{-}ply}$ we denote the subclass of $\mathbb{DG}$ of disk graphs $G_{\mathcal{D}}$ which have a representation by a τ-ply neighborhood system $\mathcal{D}$.*

The following result was proven in [147, Theorem 5.1] and [88, Theorems 4.3 and 5.1].[9]

Theorem 4.3.15 *On the class $\mathbb{DG}_{\tau\text{-}ply}$ there exists a $\sqrt{\cdot}$-separator theorem for the measure $|\cdot|$ with constants $\alpha = \frac{3}{4}$ and $\beta = O(\sqrt{\tau})$. The corresponding separation can be found in linear time.*

There is a close relation of τ-ply neighborhood systems to disks of bounded radius ratio with ϑ-precision. This relation will then establish a separator theorem on $\mathbb{DG}_{\rho,\vartheta}$.

Lemma 4.3.16 *For every $\rho \geq 1$ and $\vartheta > 0$ there exists constant τ with $\tau = O((\frac{\rho}{\vartheta})^2)$ such that $\mathbb{DG}_{\rho,\vartheta} \subseteq \mathbb{DG}_{\tau\text{-}ply}$.*

Proof: Let $\mathcal{D}$ be a collection of disks with radii in $[1, \rho]$ and ϑ-precision. A given disk $D \in \mathcal{D}$ of radius ρ and center z can be intersected only by other disks from $\mathcal{D}$ having their center in a cycle of radius 2ρ centered around z. Since $\mathcal{D}$ has ϑ-precision there can be at most τ many centers inside this cycle of radius 2ρ where τ is some constant with $\tau = O((\frac{\rho}{\vartheta})^2)$. $\square$

Theorem 4.3.15 then directly implies the following.

Theorem 4.3.17 *On the class $\mathbb{DG}_{\rho,\vartheta}$ of disk graphs with bounded radius ratio and ϑ-precision there exists a $\sqrt{\cdot}$-separator theorem for the measure $|\cdot|$ with constants $\alpha = \frac{3}{4}$ and $\beta = O(\frac{\rho}{\vartheta})$.* $\square$

This means, that we can directly apply Theorem 4.2.1 from Section 4.2.1 to the problems par-VERTEX COVER, par-INDEPENDENT SET, and par-DOMINATING SET for which we established linear problem kernels on $\mathbb{DG}_{\rho,\vartheta}$ in Corollaries 2.1.8, 4.3.11, 4.3.12 and Remark 4.3.13.

Corollary 4.3.18 (i) par-VERTEX COVER *on $\mathbb{DG}_{\rho,\vartheta}$ can be solved in time $O(2^{O(\sqrt{k})} + n)$.*[10]

[9] The result in [88, 147] was stated for the d-dimensional case.

[10] The constant hidden in the O-notation of the exponent depends on $(\frac{\rho}{\vartheta})^2$.

(ii) par-INDEPENDENT SET *on* $\mathbb{DG}_{\rho,\vartheta}$ *can be solved in time* $O(2^{O(\sqrt{k})} + n)$.[10]

(iii) par-DOMINATING SET *on* $\mathbb{DG}_{\rho,\vartheta}$ *can be solved in time* $O(2^{O(\sqrt{k})} + n)$.[10]

4.3.3.2 $\sqrt{\cdot}$-Separator Theorem on Disk Graphs with Bounded Radius Ratio

We now prove an analogue to the classical $\sqrt{\cdot}$-separator theorems for the class $\mathbb{DG}_\rho$ of disk graphs with bounded radius ratio. Note that graphs in $\mathbb{DG}_\rho$ may contain arbitrary large cliques, which means that a $\sqrt{\cdot}$-separator theorem cannot hold for the counting measure $|\cdot|$ (see Example 4.1.3 and Definition 4.1.4). This contrasts to the class of disk graphs $\mathbb{DG}_{\rho,\vartheta}$ with bounded radius ratio *and* ϑ-precision (see Theorem 4.3.17). However, if we use the (Lebesgue) measure $\mu(\cdot)$ (see Definition 4.3.5) for disk graphs with bounded radius ratio, we obtain the following result.

Theorem 4.3.19 *On the class $\mathbb{DG}_\rho$ of disk graphs with bounded radius ratio, there exists a $\sqrt{\cdot}$-separator theorem for the Lebesgue graph measure $\mu(\cdot)$.*

More precisely, there exist constants $\alpha < 1$ and $\beta > 0$ such that, for every graph $G_\mathcal{D} \in \mathbb{DG}_\rho$ with representation $\mathcal{D}$, we find three sets $\mathcal{D}_A, \mathcal{D}_S, \mathcal{D}_B \subseteq \mathcal{D}$, such that $(V_{\mathcal{D}_A}, V_{\mathcal{D}_S}, V_{\mathcal{D}_B})$ is a separation for $G_\mathcal{D}$, satisfying

(i) $\mu(\mathcal{D}_S) \leq \rho^2\beta\sqrt{\mu(\mathcal{D})}$,

(ii) $\mu(\mathcal{D}_A), \mu(\mathcal{D}_B) \leq \alpha\mu(\mathcal{D})$.

Moreover, this separation can be found in time $O(\rho^2|V_\mathcal{D}|)$.

In order to prove the Theorem, we need the following result which relates the size of the covering grid $H^\delta_\mathcal{D} = (W^\delta_\mathcal{D}, E^\delta_\mathcal{D})$ (see Definition 4.3.7) to the space occupied by the disks $\mathcal{D}$.

Lemma 4.3.20 *For any $\varepsilon > 0$ there exists $\delta > 0$ such that for any set of disks $\mathcal{D}$, each of radius at least one, it holds*[11]

$$|W^\delta_\mathcal{D}| \leq \frac{1+\varepsilon}{\delta^2}\mu(\mathcal{D}).$$

Proof: We first of all claim that if we multiply all radii of disks in $\mathcal{D}$ by an arbitrary factor $\eta \geq 1$, the total area of the new set is at most $\eta^2\mu(\mathcal{D})$. To see this, we use the theorem of Bern and Sahai [37] stating that if any set of disks is shifted in the plane by a continuous motion such that the center-to-center distance does not increase in time, then the total area of the union of disks is also non-increasing.

If we first multiply the coordinates of the centers of $\mathcal{D}$ as well as their radii by η, the new set of disks covers the space $\eta^2\mu(\mathcal{D})$. We now shift all disk centers to the original position by the continuous mapping corresponding to the continuous application of magnification factor of the range $\eta \to 1$. The claim follows by applying Bern and Sahai's theorem to this motion.[12]

[11] Note that this is a non-trivial result since $|W^\delta_\mathcal{D}|$ grows with decreasing δ.
[12] Jiří Matoušek suggested the elegant argument for this proof.

```
procedure geometric_separator
```

/* **input:** a set of disks $\mathcal{D}$ of bounded radius ratio. */
/* **output:** three sets $\mathcal{D}_S$, $\mathcal{D}_A$ and $\mathcal{D}_B$ corresponding to */
/* a separation $(V_{\mathcal{D}_A}, V_{\mathcal{D}_S}, V_{\mathcal{D}_B})$ of $G_{\mathcal{D}}$ (with properties stated in Theorem 4.3.19). */

- scale $\mathcal{D}$ such that the smallest disk has unit radius.
- fix any $\varepsilon < \frac{1}{2}$ and select δ according to Lemma 4.3.20 (e.g., $\varepsilon = \frac{1}{4}, \delta = \frac{1}{20}$) and construct the covering grid $H^{\delta}_{\mathcal{D}} = (W^{\delta}_{\mathcal{D}}, E^{\delta}_{\mathcal{D}})$ (see Definition 4.3.7)
- run the algorithm of Lipton and Tarjan (see Theorem 4.1.5) on the graph $H^{\delta}_{\mathcal{D}}$ to obtain a separation (W_A, W_S, W_B) with

  a) $|W_S| \leq \beta' \sqrt{|W^{\delta}_{\mathcal{D}}|}$ and

  b) $|W_A|, |W_B| \leq \alpha' |W^{\delta}_{\mathcal{D}}|$,

  for the constants $\beta' = 2\sqrt{2}$ and $\alpha' = \frac{2}{3}$.
- return the three sets[a]

$$\mathcal{D}_S := \mathcal{D}[\hat{N}(W_S)],^{b} \quad \mathcal{D}_A := \mathcal{D}[W_A] \setminus \mathcal{D}_S, \quad \mathcal{D}_B := \mathcal{D}[W_B] \setminus \mathcal{D}_S.$$

[a] Let $\mathcal{D}$ be a collection of disks and consider a set $S \subseteq \mathbb{R}^2$ (e.g. a set of grid vertices or a set of faces). Then $\mathcal{D}[S] = \{D \in \mathcal{D} \mid D \cap S \neq \emptyset\}$ is called the set of disks induced by S.

[b] For the definition of the face neighborhood $\hat{N}(\cdot)$, refer to Subsection 4.3.1.

Figure 4.5: Separator algorithm corresponding to Theorem 4.3.19. For an example of the construction, we refer to the right-hand diagram in Fig. 4.4.

Now, fix an arbitrary positive $\delta \leq \frac{1}{3}(\sqrt{2(1+\varepsilon)} - \sqrt{2})$ and consider the set $W^{\delta}_{\mathcal{D}}$. Each point w of $W^{\delta}_{\mathcal{D}}$, could be represented as a unique square of side size δ with w placed at the center. (These squares correspond to the grid squares shifted by $\frac{\delta}{2}$ in both coordinates.) All these new squares could be covered by the original disks if we enlarge all radii of the disks by the factor $(1 + \frac{3\sqrt{2}}{2}\delta)$. Then, applying the claim above, we get

$$\delta^2 |W^{\delta}_{\mathcal{D}}| \leq \left(1 + \frac{3\sqrt{2}}{2}\delta\right)^2 \mu(\mathcal{D}) \leq (1+\varepsilon)\mu(\mathcal{D}).$$

□

Proof (of Theorem 4.3.19):

The key idea is to apply Lipton and Tarjan's planar separator theorem to the covering grid $H^{\delta}_{\mathcal{D}} = (W^{\delta}_{\mathcal{D}}, E^{\delta}_{\mathcal{D}})$ (see Definition 4.3.7) for $\mathcal{D}$ and to construct the sets $\mathcal{D}_S$, $\mathcal{D}_A$ and $\mathcal{D}_B$ in a suitable manner from the separation obtained by Lipton and Tarjan's algorithm. The procedure which determines the sets $\mathcal{D}_S$, $\mathcal{D}_A$ and $\mathcal{D}_B$ is given in Fig. 4.5. An example for the construction is illustrated in the right-hand diagram in Fig. 4.4. We now prove that, indeed $(V_{\mathcal{D}_A}, V_{\mathcal{D}_S}, V_{\mathcal{D}_B})$ is a separation of G and that properties (i) and (ii) of the theorem for the computed sets $\mathcal{D}_S, \mathcal{D}_A$, and $\mathcal{D}_B$ hold:

$(V_{\mathcal{D}_A}, V_{\mathcal{D}_S}, V_{\mathcal{D}_B})$ *is a separation of* G*:* Showing that $(V_{\mathcal{D}_A}, V_{\mathcal{D}_S}, V_{\mathcal{D}_B})$ is a separation of $G_{\mathcal{D}}$ is equivalent to proving that $(\bigcup \mathcal{D}_A) \cap (\bigcup \mathcal{D}_B) = \emptyset$. Recall that (W_A, W_S, W_B) is the separation of $H^{\delta}_{\mathcal{D}}$ obtained by the algorithm of Lipton and Tarjan. First of all we claim that

$$\left(\bigcup \mathcal{D}_A\right) \cap W^{\delta}_{\mathcal{D}} \subseteq W_A, \quad \text{and} \quad \left(\bigcup \mathcal{D}_B\right) \cap W^{\delta}_{\mathcal{D}} \subseteq W_B. \tag{4.7}$$

To see this, note that $(\bigcup \mathcal{D}_A) \cap W_S = \emptyset$, since if there is a disk $D \in \mathcal{D}_A$ containing a point $w \in W_S$—by construction of $\mathcal{D}_S$—we had $D \in \mathcal{D}_S$. Suppose now that there is a vertex $w_B \in W_B$ which lies in a disk $D \in \mathcal{D}_A$. By definition of the set $\mathcal{D}_A$, we find a vertex $w_A \in W_A$ inside D as well. Then, there exists a path P in $H^{\delta}_{\mathcal{D}}$ which connects w_A and w_B and is completely placed inside the disk D. Since (W_A, W_S, W_B) is a separation of $H^{\delta}_{\mathcal{D}}$ there exists a vertex of W_S on P, contradicting $(\bigcup \mathcal{D}_A) \cap W_S = \emptyset$. Since $W^{\delta}_{\mathcal{D}} = W_A \cup W_S \cup W_B$, we get $(\bigcup \mathcal{D}_A) \cap W^{\delta}_{\mathcal{D}} \subseteq W_A$. The property $(\bigcup \mathcal{D}_B) \cap W^{\delta}_{\mathcal{D}} \subseteq W_B$ follows similarly.

Assume now, for contradiction, that two vertices v_A and v_B which correspond to disks $D_A \in \mathcal{D}_A$ and $D_B \in \mathcal{D}_B$ form an edge in $G_{\mathcal{D}}$. That means there exists some point z in $D_A \cap D_B$. Let $F_z \in \mathcal{F}^{\delta}$ be any of the (at most four) squares containing z, and let W_z be the four grid vertices adjacent to F_z in $H^{\delta}_{\mathcal{D}}$.

We claim that we can find two grid points $w_A \in D_A$, $w_B \in D_B$, that are in $H^{\delta}_{\mathcal{D}}$ at distance at most two. W.l.o.g. we may assume that D_A intersects at least one corner vertex w_A of F_z (otherwise we symmetrically exchange subscripts A and B). We now consider three cases (all cases are depicted in Fig. 4.6). Either D_B also intersects a corner vertex w_B of F_z (see the two left-most cases in Fig. 4.6) or D_B does not intersect any of the corner vertices of F_z (see the right-hand case in Fig. 4.6). In case D_B intersects a corner vertex w_B, this vertex either is adjacent to w_A (see the left-hand diagram in Fig. 4.6) or it lies opposite to w_A (see the middle diagram in Fig. 4.6). In any case, w_A and w_B are at distance at most two. Now consider the case, where D_B does not intersect any of the corner vertices of F_z (see the last case depicted in Fig. 4.6). In this case, it intersects one side of F_z, and since $\delta \ll 1$, it contains two grid vertices of the square sharing this side. In particular, there is one grid vertex $w_B \in D_B$ which has distance at most two to $w_A \in D_A$.

By Eq. (4.7), we have $w_A \in W_A$ and $w_B \in W_B$ and, hence, w_A and w_B must be separated by W_S. If w_A and w_B are neighbors this is a contradiction. If they have distance two in $H^{\delta}_{\mathcal{D}}$, the vertex w connecting w_A and w_B must lie in W_S. Since w lies on the boundary of F_z, we have $F_z \in \hat{N}(W_S)$. Thus, since by construction $\mathcal{D}_S := \mathcal{D}[\hat{N}(W_S)]$, we get $D_A, D_B \in \mathcal{D}_S$, a contradiction.

ad property (i): Consider a vertex $w \in W_S$ and the face neighborhood $\hat{N}(w)$ consisting of all four faces of the covering grid that are adjacent to w. We claim that all disks in $\mathcal{D}$ which intersect $\hat{N}(w)$ must lie inside a cycle of radius $(2\rho + \sqrt{2}\delta)$ centered at w. This is clear, since the covering grid has span δ and, hence, $\hat{N}(w)$ lies in a cycle of radius $\sqrt{2}\delta$ centered at w. Moreover, all disks in $\mathcal{D}$ have diameter bounded by 2ρ. Thus, a disk that intersects $\hat{N}(w)$ must lie inside a cycle of radius $(2\rho + \sqrt{2}\delta)$ centered at w.

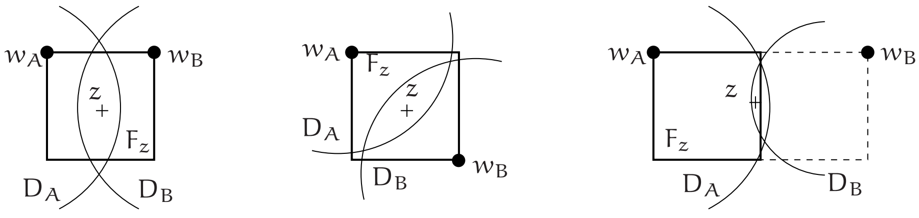

Figure 4.6: Intersecting disks select two vertices w_A and w_B from $H^\delta_{\mathcal{D}}$ at distance at most two.

This implies that

$$\begin{aligned}\mu(\mathcal{D}_S) &= \mu(\mathcal{D}[\hat{N}(W_S)]) = \mu\Big(\bigcup_{w\in W_S} \mathcal{D}[\hat{N}(w)]\Big) \qquad (4.8)\\ &\le \sum_{w\in W_S} \mu(\mathcal{D}[\hat{N}(w)]) \le \Big((2\rho+\sqrt{2}\delta)^2\pi\Big)\,|W_S| \le 5\rho^2\pi|W_S|.\end{aligned}$$

In the last step we used the following argument. If we choose $\varepsilon < \frac{1}{2}$, by the choice of δ according to Lemma 4.3.20, we have $\sqrt{2}\delta < \frac{1}{6} \le \frac{\rho}{6}$. Using property (a) of the third step of the algorithm in Fig. 4.5 and Lemma 4.3.20, we have $|W_S| \le \beta'\sqrt{|W^\delta_{\mathcal{D}}|} \le \beta'\sqrt{\frac{1+\varepsilon}{\delta^2}\mu(\mathcal{D})}$ which together with the estimate (4.8) establishes

$$\mu(\mathcal{D}_S) \le \rho^2\beta\sqrt{\mu(\mathcal{D})} \quad \text{for} \quad \beta := \frac{5\pi\beta'}{\delta}\sqrt{(1+\varepsilon)}. \qquad (4.9)$$

ad property (ii): First of all, observe that the set $\bigcup \mathcal{D}_A$ is completely covered by the square faces of the subgraph $H^\delta_{\mathcal{D}}[W_A]$, induced by the vertices of W_A. To see this, suppose there is a point $z \in D$ (for some $D \in \mathcal{D}_A$) which lies in some square $F_z \in \mathcal{F}^\delta$ of $H^\delta_{\mathcal{D}}$ but not of $H^\delta_{\mathcal{D}}[W_A]$. As above, if the four vertices adjacent to F_z host a vertex of W_B or W_S, we get $D \cap \hat{N}(W_S) \neq \emptyset$, a contradiction.

By this observation and by the fact that $|W_A| \le \alpha'|W_{\mathcal{D}}|$, we get

$$\mu(\mathcal{D}_A) \le \mu(H^\delta_{\mathcal{D}}[W_A]) \le \delta^2|W_A| \le \delta^2\alpha'|W^\delta_{\mathcal{D}}| \le \alpha'(1+\varepsilon)\,\mu(\mathcal{D}),$$

where Lemma 4.3.20 was used in the last step. Note that $\alpha := \alpha'(1+\varepsilon) < 1$ since $\varepsilon < \frac{1}{2}$.

Similarly, one proves $\mu(\mathcal{D}_B) \le \alpha'(1+\varepsilon)\,\mu(\mathcal{D})$.

running time: The running time of the algorithm is determined by the time needed to apply Lipton and Tarjan's algorithm to the graph $H^\delta_{\mathcal{D}}$ which is $O(W^\delta_{\mathcal{D}})$. Using Lemma 4.3.20, we get

$$|W^\delta_{\mathcal{D}}| \le (1+\varepsilon)\delta^{-2}\mu(\mathcal{D}) \le (1+\varepsilon)\delta^{-2}\pi\rho^2|\mathcal{D}| = O(\rho^2|\mathcal{D}|).$$

□

Observe that the choice of ε affects the tradeoff of getting small α on the one hand (it may be arbitrary close to α' as $\varepsilon \to 0$) which on the other hand causes the growth of β according to equation (4.9) as well as the growth of $|W^{\delta}_{\mathcal{D}}|$. E.g., by our choice of $\varepsilon = \frac{1}{4}$ and $\delta = \frac{1}{20}$, we get $\alpha = \alpha'(1+\varepsilon) = \frac{5}{6}$, but $\beta > 460$ and any disk of unit radius intersects at least 1250 grid points.

4.3.4 An Exact Algorithm for par-INDEPENDENT SET on Disk Graphs

As an application of the new geometric separator theorem we give an algorithm for par-INDEPENDENT SET on disk graphs $\mathbb{DG}_\rho$ of bounded radius ratio. It is known [62] that INDEPENDENT SET is NP-hard even for unit disk graphs $\mathbb{DG}_1$ by a reduction from VERTEX COVER on planar graphs with maximum degree three. The reduction in [62] can be adapted easily to obtain NP-hardness for INDEPENDENT SET on the more restricted class of unit disk graphs with 1-precision. Heuristic approaches for INDEPENDENT SET on unit disk graphs are studied in [143]. Another way to cope with this hardness was proposed by approximation theory [89, 118, 144]. Very recently, Erlebach *et al.* [89] gave a PTAS for INDEPENDENT SET on disk graphs, which is based on a sophisticated use of so-called shifting techniques as they were introduced in Baker's work [31] for deriving polynomial time approximation schemes for various planar graph problems.

Here, we are interested in *exact* solutions for the given problem. In a first subsection, we give an exact algorithm for par-INDEPENDENT SET on disk graphs of bounded radius ratio with exponential running time that is sublinear in the exponent. In a second subsection, we compare our results for disk graphs with known upper and lower bounds on planar and general graphs.

4.3.4.1 The Algorithm

We use the geometric kernelization of Subsection 4.3.2 and a divide-and-conquer approach based on the new geometric separator theorem from Subsection 4.3.3.

Theorem 4.3.21 *The* par-INDEPENDENT SET *problem on disk graphs $\mathbb{DG}_\rho$ of bounded radius ratio can be solved in time $2^{O(\sqrt{k}\log(n))}$,*[13] *where k is the size of the independent set we seek for.*

Proof: On input instance $(G_{\mathcal{D}}, k)$, in a first step, perform the geometric kernelization explained from Proposition 4.3.8. After this step, without loss of generality, we may assume that $\mu(\mathcal{D}) \leq ck$ for some constant c, depending on ρ^2.

In a second step, the divide-and-conquer procedure `indep_set` shown in Fig. 4.7 is applied to the disks $\mathcal{D}$ of the kernel $G_{\mathcal{D}}$.

Denote by $T(n, s)$ the time needed to compute `indep_set(`$\mathcal{D}$`)` of a collection of n disks $\mathcal{D}$ with $\mu(\mathcal{D}) \leq s$. Let $p(|\mathcal{D}|)$ be the polynomial time needed to compute the sets $\mathcal{D}_S, \mathcal{D}_A$, and $\mathcal{D}_B$ according to Theorem 4.3.19, and let $q(|\mathcal{D}|)$ be the polynomial time needed to perform the constructions of $\mathcal{D}'_A$ and $\mathcal{D}'_B$. Note that in $\mathcal{D}_S$ at most $\lfloor \frac{\rho^2\beta\sqrt{s}}{\pi} \rfloor$ many disks can be independent, since $\mu(\mathcal{D}_S) \leq \rho^2\beta\sqrt{s}$ and every disk has radius at least one. Hence, the total number of

[13] The constant hidden in the O-notation depends on ρ^3.

```
procedure indep_set (disks 𝒟)
/* input:   a set of disks 𝒟 of bounded radius ratio.                        */
/* output:  a maximal independent set for the corresponding disk graph G_𝒟.  */
```

- `if` ($\mathcal{D} = \emptyset$) `then` return $\emptyset$ `else`
- run `geometric_separator(`$\mathcal{D}$`)` to obtain sets $\mathcal{D}_S$, $\mathcal{D}_A$, and $\mathcal{D}_B$ (see Fig. 4.5).
- `for` all independent sets I_S of vertices in $G_{\mathcal{D}_S}$ `do`

$$\begin{aligned} \mathcal{D}'_A &:= \mathcal{D}_A \setminus \{ D \in \mathcal{D}_A \mid \exists D' \in \mathcal{D}[I_S] : D \cap D' \neq \emptyset \} \\ \mathcal{D}'_B &:= \mathcal{D}_B \setminus \{ D \in \mathcal{D}_B \mid \exists D' \in \mathcal{D}[I_S] : D \cap D' \neq \emptyset \} \\ I'_S &:= I_S \cup \texttt{indep\_set}(\mathcal{D}'_A) \cup \texttt{indep\_set}(\mathcal{D}'_B) \end{aligned}$$

- return $(I'_S)_{\mathrm{opt}}$, for which

$$|(I'_S)_{\mathrm{opt}}| = \min \{ |I'_S| \mid I_S \text{ is an independent set in } G_{\mathcal{D}_S} \}$$

Figure 4.7: Divide-and-conquer algorithm for INDEPENDENT SET on disk graphs of bounded radius ratio based on the new geometric separator theorem. Note that for a fixed independent set I_S of $G_{\mathcal{D}_S}$, no further disk from $\mathcal{D}_A$ and $\mathcal{D}_B$ intersecting disks from $\mathcal{D}[I_S]$ can be chosen. This explains the construction of $\mathcal{D}'_A$ and $\mathcal{D}'_B$.

independent sets in $G_{\mathcal{D}_S}$ is upper bounded by

$$\sum_{i=0}^{\lfloor \frac{\rho^2 \beta \sqrt{s}}{\pi} \rfloor} \binom{n}{i} \leq n^{\widehat{\beta}\sqrt{s}},$$

where $\widehat{\beta}$ is some constant, depending on ρ^2.[14] Then, the recursion we have to solve in order to compute an upper bound on $T(n, s)$ reads as follows:

$$T(n,s) \leq p(n) + n^{\widehat{\beta}\sqrt{s}} \cdot q(n) \cdot 2\,T(n, \alpha s).$$

Hence, for n large enough, and a suitable constant $\widetilde{\beta}$ we have

$$\begin{aligned} T(n,s) &\leq n^{\widetilde{\beta}\sqrt{s}} \cdot T(n, \alpha s) \leq \prod_{i=0}^{\log_{\frac{1}{\alpha}}(s)} n^{\widetilde{\beta}\sqrt{\alpha^i s}} \cdot T(n,1) \\ &\leq n^{\widetilde{\beta}\sqrt{s}(\sum_{i=0}^{\infty} \alpha^{\frac{i}{2}})} \cdot T(n,1) = n^{\frac{\widetilde{\beta}\sqrt{s}}{1-\sqrt{\alpha}}} \cdot T(n,1). \end{aligned}$$

Note that $T(n, 1)$ is constant, since $\mu(\mathcal{D}) \leq 1$ implies $\mathcal{D} = \emptyset$, because for every disk D we have $\mu(D) \geq \pi$. By plugging in the values $n = |\mathcal{D}|$ and $s = ck$, we obtain the running time

[14] Observe that this is an argument which solely works for INDEPENDENT SET and cannot be carried over to other problems. E.g., in the case of DOMINATING SET, the fact that $\mu(\mathcal{D}_S) \leq c\sqrt{s}$ does not imply an upper bound of the form $n^{c'\sqrt{s}}$ on the total number of dominating sets in $G_{\mathcal{D}_S}$.

graph class	*(classical) complexity*	*parameterized complexity*
general graphs	$2^{0.25\,n}$ [174, 175] lower bound: $2^{\Omega(n)}$ [119]	W[1]-complete [81, 82]
disk graphs $\mathbb{DG}_\rho$	$2^{O(\sqrt{n}\log(n))}$ [Rem. 4.3.22]	$2^{O(\sqrt{k}\log(n))}$ [Thm. 4.3.21] open: FPT or W[1]-hard ?
disk graphs $\mathbb{DG}_{\rho,\vartheta}$ (with ϑ-precision)	$2^{O(\sqrt{n})}$ [Cor. 4.3.18, $k = n$]	$O(2^{O(\sqrt{k})} + n)$ [Cor. 4.3.18] hence: FPT
planar graphs	$2^{O(\sqrt{n})}$ [138] lower bound: $2^{\Omega(\sqrt{n})}$ [Thm. 4.2.10, Prop. 2.1.4]	$O(2^{O(\sqrt{k})} + n)$ [Cor. 4.2.2] hence: FPT

Table 4.2: Relating our results on par-INDEPENDENT SET on disk graphs to known results for general graphs and for planar graphs. (Lower bounds are under the assumption that 3 SAT $\notin$ DTIME($2^{o(n)}$), where n is the number of variables of a 3 SAT formula.) For disk graphs $\mathbb{DG}_{\rho,\vartheta}$ and planar graphs, similar results also hold true for par-VERTEX COVER and par-DOMINATING SET (see Corollaries 4.2.2 and 4.3.18).

$2^{O(\sqrt{k}\log(n))}$ as we have claimed. Since $\widetilde{\beta}$ and c both depend on ρ^2, the constant hidden in the O-notation depends on ρ^3. □

Remark 4.3.22 In the worst-case $k = n$, we have a time $2^{O(\sqrt{n}\log(n))}$ algorithm.

4.3.4.2 Summary and Comparison

We briefly summarize our results on the parameterized and classical complexity of the INDEPENDENT SET problem for disk graphs and give a comparison to known results on general and planar graphs (which are equivalent to the class of coin graphs); see Table 4.2 for an overview.

In classical complexity study, the best known algorithm for INDEPENDENT SET on general graphs running in time $2^{0.276n}$ is due to Robson [174] which he recently improved to $2^{0.25n}$ [175].

Impagliazzo, Paturi, and Zane [119] investigated the syntactic class SNP (as originally defined by Papadimitriou and Yannakakis [162]) under so-called Subexponential Reduction Families (SERF). They showed that the existence of a "subexponential" algorithm for any problem that is SNP-hard under SERF-reduction implies that every problem in SNP has a "subexponential" algorithm [119, Lemma 9]. In particular, since INDEPENDENT SET and 3 SAT are proven to be SNP-complete under SERF-reduction [119, Theorem 3], it follows that INDEPENDENT SET $\in$ DTIME($2^{o(n)}$) is impossible unless 3 SAT $\in$ DTIME($2^{o(n)}$), n being the number of variables in the latter case.[15] This fact, i.e., that 3 SAT can be solved in exponential time with a sublinear exponent, is generally considered to be very unlikely.[16]

[15] Basically, the implication INDEPENDENT SET $\in$ DTIME($2^{o(n)}$) $\Rightarrow$ 3 SAT $\in$ DTIME($2^{o(n)}$) can be proven in a very similar way to the proof of Proposition 4.2.4 combined with Lemma 4.2.5.

[16] So far, the best known algorithm for 3 SAT has running time $O(1.481^n)$[115].

From parameterized complexity theory, we know that par-INDEPENDENT SET is $W[1]$-complete on general graphs [81, 82].

If restricted to planar graphs, Lipton and Tarjan applied their well-known planar separator theorem [137] to get an algorithm with running time $2^{O(\sqrt{n})}$ (also see Proposition 4.1.15). In parameterized complexity study, par-INDEPENDENT SET on planar graphs is in FPT, and for the (asymptotically) best known algorithm we get running time $O(2^{O(\sqrt{k})} + n)$, the exponential term being sublinear in k (see Corollary 4.2.2). Moreover, recall that this algorithm matches the lower bound given in Theorem 4.2.10, which states that there is no fixed-parameter algorithm for par-INDEPENDENT SET with running time $2^{o(\sqrt{k})} n^{O(1)}$ unless $3\,\text{SAT} \in \text{DTIME}(2^{o(n)})$, n being the number of variables. In addition, this result provides a lower bound of $2^{\Omega(\sqrt{n})}$ for INDEPENDENT SET on planar graphs under the hypothesis that $3\,\text{SAT} \notin \text{DTIME}(2^{o(n)})$: an algorithm of running time $2^{o(\sqrt{n})}$ for INDEPENDENT SET on planar graphs in combination with the linear problem kernel reduction (see Proposition 2.1.4), would lead to an algorithm for par-INDEPENDENT SET on planar graphs that would be better than the given relative lower bound.

The results that could be established for planar graphs also hold for disk graphs $\mathbb{DG}_{\rho,\vartheta}$ with bounded radius ratio and ϑ-precision. Similarly to the planar case, for par-INDEPENDENT SET on $\mathbb{DG}_{\rho,\vartheta}$, we derived an algorithm of running time $O(2^{O(\sqrt{k})} + n)$ (see Corollary 4.3.18.(ii)). Letting $k = n$, we obtain a time $2^{O(\sqrt{n})}$ algorithm.

On the class $\mathbb{DG}_{\rho}$ of disk graphs with bounded radius ratio, applying our new geometric separator theorem lead to an algorithm of running time $2^{O(\sqrt{k}\log(n))}$ (see Theorem 4.3.21), which translates to a time $2^{O(\sqrt{n}\log(n))}$ algorithm if $k = n$ (see Remark 4.3.22). The exponent $O(\sqrt{n}\log(n))$ is a sublinear term and, hence, cannot be achieved for general graphs, unless $3\,\text{SAT} \in \text{DTIME}(2^{o(n)})$, n being the number of variables [119]. We leave it as an open problem whether par-INDEPENDENT SET on $\mathbb{DG}_{\rho}$ or even on general disk graphs is in FPT or complete for the class $W[1]$.

Finally, we want to mention that completely independent of our work, Lev-Tov and Peleg [136] gave (among others) exact algorithms for the INDEPENDENT SET problem on unit disk graphs with similar running times. Namely, they gave a time $2^{O(\sqrt{n}\log(n))}$ algorithm for unit disk graphs and a time $2^{O(\sqrt{n})}$ algorithm if in addition the centers of the unit disks are required to sit on a given grid of span δ. Our results generalize these results with respect to various aspects: Firstly, our algorithms are stated in the parameterized complexity setting and our results, for the classical complexity setting, are only derived as sideproduct. It might be—as in the case of a large clique—that even though n could be huge, the size of the largest independent set of a disk graph (i.e., the parameter k) can be very small. In this sense, a time $n^{O(\sqrt{n})}$ algorithm is inferior to an $n^{O(\sqrt{k})}$ algorithm. Secondly, our results are given for a much broader class of graphs than the ones in [136]. Lev-Tov and Peleg restricted their studies to the case of *unit* disk graphs, whereas our setting allows disks of *distinct* radii (as long as the ratio of largest and smallest radii is bounded). Their $2^{O(\sqrt{n})}$ only holds if the centers of the unit disks are assumed to sit on a given grid of span δ, which is more restrictive than the class of ϑ-precision graphs considered in our setting.

Chapter 5

Tree Decomposition Based Algorithms

The notions of "tree decomposition" and "treewidth" have their origin in the deep theoretical work of Robertson and Seymour on graph minors. From an algorithmic point of view, tree decompositions provide an interesting concept for designing fixed-parameter algorithms. Typically, treewidth based algorithms proceed according to the following scheme in two stages:

Phase 1: Find a tree decomposition of bounded width for the input graph, and then

Phase 2: solve the problem using dynamic programming approaches on the tree decomposition.

After providing some background on tree decompositions in Section 5.1, we take a closer look at both stages.

As to the first phase, it is generally considered to be a hard task to compute a tree decomposition of provably "small" width quickly. In Section 5.2, we introduce the abstract notion of the so-called "Layerwise Separation Property" (LSP) which holds for a broad class of parameterized problems including, e.g., par-VERTEX COVER, par-INDEPENDENT SET, par-DOMINATING SET, and various variations of these problems. For such LSP-problems, we develop a general methodology that allows the fast construction of tree decompositions of width bounded by $O(\sqrt{k})$, where k is the problem parameter.

As to the second phase, the running time of the dynamic programming typically is polynomial or even linear in the size of the input graph, but exponential in the width of the given tree decomposition. Hence, the latter turns out to be the computational bottleneck. For our threesome VERTEX COVER, INDEPENDENT SET, DOMINATING SET, the best running time behavior so far was obtained by Telle and Proskurowski [183, 184]. In Section 5.3, we revisit their work, and demonstrate how their algorithms can be improved significantly by an argument on a certain "monotonicity" in the table updating process during dynamic programming.

Finally, the results are put together in Section 5.4 to derive time $O(2^{O(\sqrt{k})} n)$ algorithms for LSP-problems, where the constant hidden in the exponent can be computed by various problem specific numbers. In combination with problem kernel reduction most of the given algorithms can be sped up to a running time of the form $2^{O(\sqrt{k})} + n^{O(1)}$.

5.1 Background

We shortly review the notion of "tree decompositions" together with the graph parameter "treewidth." Besides, we introduce the so-called "layer decomposition" for planar graphs.

5.1.1 Tree Decompositions and Treewidth

Since Robertson and Seymour defined the graph parameter treewidth together with the associated graph structure tree decomposition in their seminal work [171], this notion played an important role in both graph theory, as well as algorithm theory.

5.1.1.1 Definition and First Properties

Informally speaking, the treewidth measures how "tree-like" a graph is. For a detailed introduction to tree decompositions we refer to [128] and to the very good survey [45].

Definition 5.1.1 *Let $G = (V, E)$ be a graph. A* tree decomposition *$\mathcal{X}$ of G is a pair $\langle \{ X_i \mid i \in V(T) \}, T \rangle$, where each X_i is a subset of V, called a* bag*, and T is a tree with the elements of $V(T)$ as nodes. The following three properties must hold:*

(i) $\bigcup_{i \in I} X_i = V$;

(ii) *for every edge $\{u, v\} \in E$, there is an $i \in V(T)$ such that $\{u, v\} \subseteq X_i$;*

(iii) *for all $i, j, k \in V(T)$, if j lies on the path between i and k in T, then $X_i \cap X_k \subseteq X_j$.*

The width $\mathrm{tw}(\mathcal{X})$ *of $\mathcal{X}$ is defined to be the size of the largest bag minus one, i.e.,*

$$\mathrm{tw}(\mathcal{X}) := \max\{ |X_i| \mid i \in V(T) \} - 1.$$

The treewidth $\mathrm{tw}(G)$ *of a graph G is the minimum ℓ such that G has a tree decomposition of width ℓ.*

It is not hard to verify that the last property (iii) is equivalent to the following:

(iii)' *for every vertex $v \in V$ the graph $T_v \subseteq T$ induced by all nodes i which contain v in their bag X_i is a tree.*

An example of a graph together with a tree decomposition is given in Fig. 5.1. Using properties (ii) and (iii)', it is not difficult to derive the lower bound $\mathrm{tw}(G) \geq \mathrm{cl}(G) - 1$, where $\mathrm{cl}(G)$ denotes the size of the largest clique in a graph G.

A tree decomposition with a particularly simple structure is given by the following.

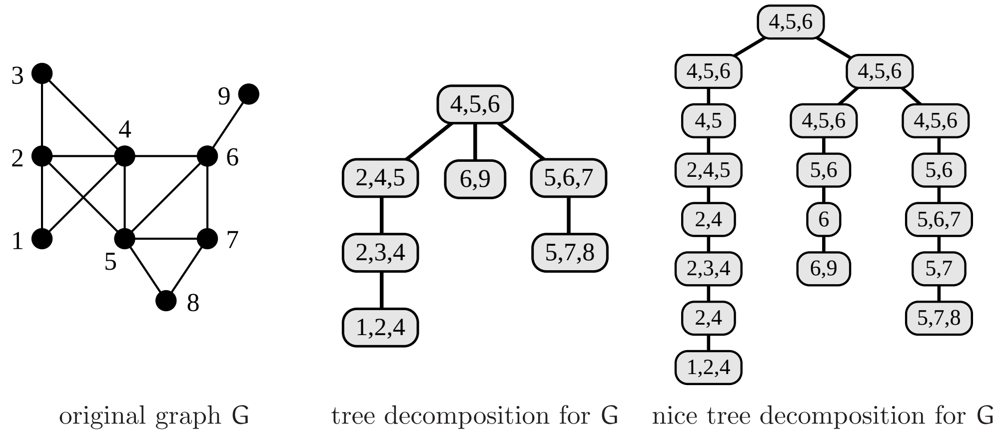

Figure 5.1: The diagram in the middle shows a tree decomposition of width two for the graph G in the left-hand diagram. Since G contains a clique of size three, this tree decomposition has the smallest possible width. The right-hand diagram shows a nice tree decomposition for G of the same width.

Definition 5.1.2 *A tree decomposition* $\langle \{X_i \mid i \in V(T)\}, T\rangle$ *is called a* nice tree decomposition *if the tree* T *is rooted and the following conditions are satisfied:*

(i) *Every node of the tree* T *has at most two children.*

(ii) *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*).*

(iii) *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*).*

It is not hard to transform a given tree decomposition into a nice tree decomposition. More precisely, the following result holds (see [128, Lemma 13.1.3]).

Lemma 5.1.3 *Given a tree decomposition of a graph* G *that has width* ℓ *and* $O(n)$ *nodes, where* n *is the number of vertices of* G. *Then, we can find a nice tree decomposition of* G *that has also width* ℓ *and* $O(n)$ *nodes in time* $O(n)$. □

Fig. 5.1 shows an example that illustrates how a given tree decomposition can be turned into a nice tree decomposition.

We finish this paragraph by informally introducing the *robber-cop game* on a graph which gives us an alternative characterization of the treewidth. This characterization is is due to Seymour and Thomas [176]. The game is played by two players; one player being a robber who can—at any time—move along the edges of a graph at arbitrary speed and one player controlling the moves of ℓ cops who either sit on a vertex or in a helicopter each by which a cop can move to an arbitrary other vertex. The robber may not pass through a vertex occupied by a cop and the aim of the player controlling the moves of the cops is to land a cop on the vertex on which the robber sits. For a precise definition of the game we refer to [176], where the authors also prove the following min-max theorem.

Proposition 5.1.4 *A graph G has treewidth at most ℓ if and only if $\ell + 1$ cops have a winning strategy in the robber-cop game on G.* □

There is a serious application of this seemingly playful result.

Corollary 5.1.5 *For the complete grid $G = P_m \times P_m$ with $m \times m$ vertices, we have* $\mathrm{tw}(G) = m$.

Proof (Sketch): It is not hard to see that $m+1$ cops do have a winning strategy in the robber-cop game on G, however, using m cops the robber always has an escape strategy. Proposition 5.1.4 then yields the claim. □

5.1.1.2 The Importance of Tree Decompositions

The importance of tree decompositions from a graph-theoretical point of view. The notion of tree decompositions played a central role in a long series of papers of Robertson and Seymour proving Wagner's conjecture which states that every minor closed family $\mathbb{G}$ of graphs admits a finite obstruction set, i.e., a finite set $\mathrm{Forb}(\mathbb{G})$ of graphs, such that $G \in \mathbb{G}$ if and only if G does not contain any graph from $\mathrm{Forb}(\mathbb{G})$ as a minor (see [172]).

The following lemma shows that, for every ℓ, the class of graphs $\mathbb{G}_{\mathrm{tw}\leq\ell}$ of treewidth bounded by ℓ is minor closed.

Lemma 5.1.6 *If H is a minor of some graph G, then* $\mathrm{tw}(H) \leq \mathrm{tw}(G)$.

Proof: If H is a subgraph of G and $\langle \{X_i \mid i \in V(T)\}, T \rangle$ is a tree decomposition for G, then it is easy to see that $\langle \{V(H) \cap X_i \mid i \in V(T)\}, T \rangle$ is a tree decomposition for H. Now suppose there is a vertex $v \in V(H)$ which was obtained by contracting the edge $\{v_1, v_2\} \in E(G)$, then a tree decomposition of G can be turned into a tree decomposition of H by replacing all occurrences of v_1 and v_2 in bags X_i with the vertex v. □

A first and decisive step in Robertson and Seymour's theorem was to prove Wagner's conjecture for the class $\mathbb{G}_{\mathrm{tw}\leq\ell}$. The finite obstruction sets $\mathrm{Forb}(\mathbb{G}_{\mathrm{tw}\leq\ell})$ are *explicitly* known only for the cases $\ell = 1, 2, 3$ [26].

The importance of tree decompositions from an algorithmic point of view. Many in general NP-complete graph problems do have polynomial or even linear time solving algorithms when the underlying graph has a tree decomposition of width bounded by a constant. In fact, Courcelle's theorem [67] states that every language that is expressible in so-called monadic second order logic can be solved in polynomial time when given a tree decomposition of the input graph of width bounded by a constant. Note that the running time is exponential in the width of the given tree decomposition.

Hence, the limiting factor in most of these algorithms, is the *efficient* construction of tree decompositions with *small* width. When given a graph G and an integer ℓ, the problem to determine whether the treewidth of G is at most ℓ, is NP-complete [23]. When the parameter ℓ is a fixed constant, however, a lot of work was done on polynomial time solutions, culminating in Bodlaender's linear time algorithm [43]—with a running time $O(2^{\Theta(\ell^3)}n)$ that (even for very small values of ℓ) still seems much too large for practical purposes. Even the time $O(n^{\ell+1})$ algorithm of Arnborg *et al.* [23] is more practical. That is why also heuristic approaches (see [131] for an up-to-date account), data reduction (see [86]), or approximation results (see, e.g., [46, 49, 70, 167]) for constructing tree decompositions are in use. From the viewpoint of approximation, an $O(\log(n))$-approximation algorithm is known [46] for treewidth on general graphs. An open problem is whether treewidth can be approximated by a constant factor in polynomial time. A 1.5-approximation algorithm (with polynomial running time) has recently been proven [70] on the class of graphs that exclude a single-crossing graph, i.e., a graph that can be embedded in the plane with a single edge-crossing.

The main contribution of this chapter will be to give new upper bounds for the treewidth of a planar graph of the form $\mathrm{tw}(G) = O(\sqrt{k})$, where k is a graph problem parameter, such as the vertex cover number, the domination number, or the stability number. Moreover, we will see how to construct the corresponding tree decompositions very quickly, i.e., in time $O(\sqrt{k}\,n)$.

5.1.2 Layer Decompositions

We now introduce a decomposition of the vertices of a plane graph (G, ϕ), according to the level of the "layer" in which they appear in an embedding ϕ. We will study the relation of this so-called "layer decomposition" and tree decompositions in Subsection 5.2.2.

The notion of graph layers was first introduced in Baker's influential work [31].

Definition 5.1.7 *Let* $(G = (V, E), \phi)$ *be a plane graph.*

(i) *The* layer decomposition *of* (G, ϕ) *is a disjoint partition of the vertex set* V *into sets* $L_1, \ldots, L_r$*, which are recursively defined as follows:*

- L_1 *is the set of vertices on the exterior face of* G*.*
- L_i *is the set of vertices on the exterior face of* $G[V - \bigcup_{j=1}^{i-1} L_j]$ *for all* $i = 2, \ldots r$*.*

We denote the layer decomposition of (G, ϕ) *by* $\mathcal{L}(G, \phi) := (L_1, \ldots, L_r)$*.*[1]

(ii) *The set* L_i *is called the* ith layer *of* (G, ϕ)*.*

(iii) *The (uniquely defined) number* r *of different layers is called the* outerplanarity *of* (G, ϕ)*, denoted by* $\mathrm{out}(G, \phi) := r$*.*

[1] Due to technical reasons, for a layer-decomposition $\mathcal{L}(G, \phi) := (L_1, \ldots, L_r)$, we set $L_i := \emptyset$ for all indices $i < 1$ and $i > r$.

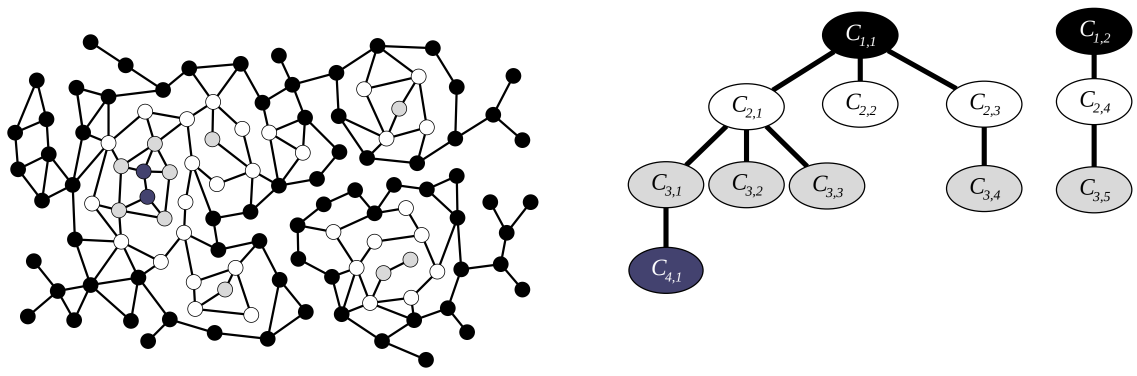

Figure 5.2: The layer decomposition forest of a plane graph with four layers.

(iv) *We define* $\mathrm{out}(G)$ *to be the smallest outerplanarity possible among all plane embeddings, i.e., minimizing over all plane embeddings* ϕ *of* G *we set*

$$\mathrm{out}(G) := \min_{\phi} \mathrm{out}(G, \phi).$$

Taking into account that each layer L_i may consist of different components, we observe that the layer decomposition in fact has a forest structure. We may define the *layer decomposition forest* as follows (Fig. 5.2 provides an example for a layer decomposition forest.): For each layer with vertex set L_i, suppose the connected components of the subgraph of $G[L_i]$ induced by L_i have vertex sets $C_{i,1}, \ldots, C_{i,\ell_i}$, i.e., $L_i = \bigcup_{j=1}^{p_i} C_{i,j}$. The nodes of the layer decomposition forest will be the vertex sets $C_{i,j}$, with $1 \leq i \leq q$ and $1 \leq j \leq p_i$, which will be called *layer component*. Two layer component nodes will be adjacent in the layer decomposition forest if they contain vertices that are adjacent. This means that a layer component node $C_{i,j}$ can only be adjacent to layer component nodes of the form $C_{i-1,j'}$ or $C_{i+1,j''}$; if $C_{i,j}$ is adjacent to $C_{i-1,j'}$, then the vertices of $C_{i,j}$ lie within the area formed by the subgraph induced by $C_{i-1,j'}$. Note that the layer component nodes on the i'th level of the forest correspond to the layer components of the form $C_{i,j}$. Each layer component node $C_{1,j}$ will be the root of a tree in the forest, so we will have one tree per connected component of G, representing the exterior vertices of the component.

Computing the layers of a plane graph can be done efficiently.

Lemma 5.1.8 *Given a plane graph* $(G = (V, E), \phi)$*, then the layer decomposition can be computed in time* $O(|V|)$.

Proof: The proof is deferred to the Appendix at the end of the chapter. □

Remark 5.1.9 For a given planar graph G, an embedding ϕ_0 with $\mathrm{out}(G, \phi_0) = \mathrm{out}(G)$, i.e., an embedding which minimizes the number of layers, can be computed in polynomial time [40]. Our algorithms work, however, for *any* planar embedding of G.

5.2 Constructing Tree Decompositions for Planar Graphs

In this section, we are concerned with "Phase 1" of the general scheme for tree decomposition based algorithms as presented at the beginning of this chapter. We present a general methodology how to construct tree decompositions for planar graphs with a guaranteed upper bound on the width of the constructed decomposition that is sublinear in a given graph parameter. This method uses a combination of graph separation (as discussed in Subsection 5.2.1) on the one hand, and known results for graphs of bounded outerplanarity (as discussed in Subsection 5.2.2) on the other hand. Moreover, we introduce the so-called "Layerwise Separation Property" (see Subsection 5.2.3) for a parameterized problem on planar graphs which guarantees that our approach will work. Besides, we give various examples of problems having this property. The methodology itself is presented in Subsection 5.2.4.

In the sequel, we follow parts of [4, 8] (also refer to [16]).

5.2.1 Separators and Treewidth

There is a close relation between tree decompositions and graph separators (as discussed in Subsection 4.1.1). On the one hand, a tree decomposition of a graph G induces in a very natural way (small) separators of G. On the other hand, in the reverse direction, graph separation can be used as a tool to construct tree decompositions. We discuss both issues in this Subsection.

Separators obtained from a tree decomposition. A tree decomposition has the property that the intersection of two adjacent bags is a separator. The following lemma is easy to see using properties (ii) and (iii) of Definition 5.1.1.

Lemma 5.2.1 *Let* $\langle \{X_i \mid i \in V(T)\}, T \rangle$ *be a tree decomposition of* G*. Consider an edge* $\{i_1, i_2\}$ *of* T *and let* T_1, T_2 *be the components of* $T - \{i_1, i_2\}$*. Then* $S := X_{i_1} \cap X_{i_2}$ *separates* $(\bigcup_{j \in V(T_1)} X_j) \setminus X_{i_1}$ *from* $(\bigcup_{j \in V(T_2)} X_j) \setminus X_{i_2}$. □

In particular, every bag X_i where $i \in V(T)$ is not a leaf node, yields a separator of the tree.

Using separators to construct a tree decomposition. Here, the main idea is to use small separators of the graph and to merge the tree decompositions of the resulting subgraphs. For any given separator splitting a graph into different components, we obtain a simple upper bound for the treewidth of this graph which depends on the size of the separator and the treewidth of the resulting components.

Proposition 5.2.2 *If a connected graph can be decomposed into components of treewidth of at most* t *by means of a separator of size* s*, then the whole graph has treewidth of at most* $t + s$*.*

Proof: The separator splits the graph into different components. Suppose we are given the tree decompositions of these components of width at most t. The goal is to construct a tree decomposition for the original graph. This can be achieved by firstly merging the separator to every bag in each of these given tree decompositions. In a second step, add some arbitrary connections preserving acyclicity between the trees corresponding to the components. It is straightforward to check that this forms a tree decomposition of the whole graph of width at most $t+s$. □

For plane graphs, there is an iterated version of this observation, which will be used for the construction of tree decompositions of small width in the later sections.

Proposition 5.2.3 *For a plane graph* (G,ϕ) *be a plane with layer decomposition* $\mathcal{L}(G,\phi) = (L_1,\ldots,L_r)$*, let*

$$\mathcal{L}_i = \{L_{j_i}, L_{j_i+1}, \ldots, L_{j_i+n_i}\},$$

where $1 \le i \le \ell$*, be sets of consecutive layers such that* $\mathcal{L}_i \cap \mathcal{L}_{i'} = \emptyset$ *for all* $i \neq i'$*.*

Suppose G *can be decomposed into components, each of treewidth of at most* t*, by means of separators* $S_1,\ldots,S_\ell$*, where* $S_i \subseteq \bigcup_{L\in\mathcal{L}_i} L$*,* $1 \le i \le \ell$*.*

Then, G *has treewidth of at most* $t+2s$*, where* $s = \max_{i=1,\ldots,\ell} |S_i|$*.*

Proof: The proof again uses the merging-technique illustrated in Proposition 5.2.2: Suppose, w.l.o.g., the sets $\mathcal{L}_i$ appear in successive order, i.e., $j_i < j_{i+1}$. For each $i = 0,\ldots,\ell$, consider the component G_i of treewidth at most t which is cut out by the separators S_i and S_{i+1} (by default, we set $S_0 = S_{\ell+1} = \emptyset$). We add S_i and S_{i+1} to every node in a given tree decomposition of G_i. In order to obtain a tree decomposition of G, we successively add an arbitrary connection between the trees T_i and T_{i+1} of the so-modified tree decompositions that correspond to the subgraphs G_i and G_{i+1}. □

5.2.2 Outerplanarity and Treewidth

The result presented in this subsection concerns the relation between treewidth and outerplanarity. An upper bound on the treewidth of the form $\mathrm{tw}(G) \le 3\cdot\mathrm{out}(G) - 1$ for planar graphs can be found in [45, Theorem 83] (also, see [134, Table 2 in Chapter 10]). Here, we sketch a *constructive* proof of the result, since it is essential for us, that a corresponding treewidth can be found quickly. To be more precise, partly following [4, 45], we show:

Theorem 5.2.4 *Let* $(G=(V,E),\phi)$ *be a plane graph with* $r := \mathrm{out}(G,\phi)$ *and* n *vertices. Then, a tree decomposition* $\langle\{X_i \mid i \in V(T)\}, T\rangle$ *of* G*, with width at most* $3r-1$ *and with at most* $2n-1$ *nodes, can be found in time* $O(r\,n)$*.*

Let us first of all describe a construction scheme that yields—in a very simple way—a tree decomposition for a graph that is given together with a maximal spanning forest. The width of the resulting tree decomposition will depend on the following numbers.

Definition 5.2.5 *For a graph* $G=(V,E)$*, and a forest* $T=(V,F)$ *that is a subgraph of* G*, define the* edge remember number $\mathrm{er}(G,T,e)$ *of an edge* $e \in F$ *(with respect to* G *and* T*) as the number of edges* $\{v,w\} \in E \setminus F$ *such that there is a simple path in* T *from* v *to* w *that uses* e*. The edge* *remember number of* T *(with respect to* G*) is*

$$\mathrm{er}(G,T) := \max_{e\in F} \mathrm{er}(G,T,e).$$

The vertex remember number $\mathrm{vr}(G,T,v)$ *of a vertex* $v \in V$ *(with respect to* G *and* T*) is the number of edges* $\{u,w\} \in E \setminus F$ *such that there is a simple path in* T *from* u *to* w *that uses* v*. The vertex remember number of* T *(with respect to* G*) is*

$$\mathrm{vr}(G,T) := \max_{v\in V} \mathrm{vr}(G,T,v).$$

With this notion at hand, the following result follows [42, Theorem 71]. For the sake of completeness of the whole construction, we outline the easy proof.

Proposition 5.2.6 *Let* $T = (V,F)$ *be a maximal spanning forest for the graph* $G = (V,E)$*. Then, a tree decomposition with width at most* $\max(\mathrm{vr}(G,T), \mathrm{er}(G,T)+1)$ *and at most* $2n-1$ *nodes can be determined in* $O(\mathrm{vr}(G,T)\cdot n)$ *time,* n *being the number of vertices in* G*.*

Proof: Our aim is to construct a tree decomposition $\langle \{X_i \mid i \in V(T')\}, T'\rangle$ of G. Let $T' = (V \cup F, F')$ with $F' = \{\{v,e\} \mid v \in V, e \in F, \exists w \in V : e = \{v,w\}\}$ be the tree obtained by subdividing every edge of T. Observe that T' has at most $2n-1$ nodes. The bags X_i for $i \in V(T') = V \cup F$ are obtained as follows. For every $v \in V$, add v to X_v. For every $e = \{v,w\} \in F$, add v and w to X_e. Now, for every edge $e = \{v,w\}$ in E but not in F, add v to all sets X_u and X_e, with $u \in V$ or $e \in F$ on the path from v to w in T.

Using standard graph algorithmic techniques, the path between two vertices in a tree can be found in time proportional to the length of that path; since each vertex in T can contribute to at most $\mathrm{vr}(G,T)$ such paths, the running time is bounded by $O(\mathrm{vr}(G,T)\cdot n)$.

It is easy to check that this indeed yields a tree decomposition. Its bags have size $|X_v| \leq 1+\mathrm{vr}(G,T)$ (for all $v \in V$) and $|X_e| \leq 2+\mathrm{er}(G,T)$ (for all $e \in E$). Hence, the resulting treewidth is at most $\max(\mathrm{vr}(G,T), \mathrm{er}(G,T)+1)$. □

In order to give an algorithm that proves Theorem 5.2.4, it remains to show that we can find a suitable maximal spanning tree for G (see [45, Lemma 80]).

Proposition 5.2.7 *Let an* r*-outerplanar graph* $G = (V,E)$ *with degree bounded by three be given together with an* r*-outerplanar embedding. Then, there exists a maximal spanning forest* T *for* G*, such that* $\mathrm{er}(G,T) \leq 2r$ *and* $\mathrm{vr}(G,T) \leq 3r-1$*. The construction of the spanning forest can be done in time* $O(r\,n)$*, where* n *is the number of vertices in* G*.*

Proof: The construction of a maximal spanning forest T for G is done inductively along the different layers proceeding from inside towards the exterior.

Observe that when removing all *edges* on the exterior face of an s-outerplanar graph of maximum degree three, we obtain an $(s-1)$-outerplanar graph, when $s > 1$.

Thus, we can partition the edges into $r+1$ sets $E_1, \ldots, E_{r+1}$, with E_1 the edges on the exterior face, and E_i the edges on the exterior face when all edges in $E_1 \cup \ldots \cup E_{i-1}$ are removed. Similar to Lemma 5.1.8, using the dual, this partition can be computed in $O(n)$ time.

Now, we form a sequence of forests. We start with forest T_{r+1}, which consists of all edges in E_{r+1}. (Note that these are the interior edges of an outerplanar graph that has, by assumption, maximum degree 3; so T_{r+1} is acyclic.) When we have T_i, $1 < i \leq r+1$, we form T_{i-1} in the following way: add a maximal set of edges from E_{i-1} to T_i such that no cycles are formed. Note that in this way, each T_i is a maximal spanning forest of the subgraph formed by the edges in $E_i \cup \ldots \cup E_{r+1}$; we call this subgraph G_i. Let $T := T_1$. We claim that for every $1 < i < r$, the following upper bounds hold

$$\begin{aligned} \mathrm{er}(G_{i-1}, T_{i-1}) &\leq \mathrm{er}(G_i, T_i) + 2, \qquad \mathrm{er}(G_r, T_r) = 2, \\ \mathrm{vr}(G_{i-1}, T_{i-1}) &\leq \mathrm{vr}(G_i, T_i) + 3, \qquad \mathrm{vr}(G_r, T_r) = 2. \end{aligned}$$

These statements can be derived by induction using a straightforward combinatorial analysis of the construction (see [45, Lemma 80] for details). Hence, we get $\mathrm{er}(G, T) = \mathrm{er}(G_1, T_1) = 2r$ and $\mathrm{vr}(G, T) = \mathrm{vr}(G_1, T_1) = 3r - 1$

Concerning the claimed running time, it is not hard to see that one such step can be done in $O(n)$ time; as we do at most r such steps, the time to build $T := T_1$ becomes $O(r\,n)$. □

Now, we can complete the proof of Theorem 5.2.4.

Proof (of Theorem 5.2.4): For the construction of the desired tree decomposition we proceed in the following steps for a given plane graph (G, ϕ) with $r = \mathrm{out}(G, \phi)$.

- Embed the graph G in an r'-outerplanar (with $r' \leq r$) graph H of degree bounded by three, such that G is a minor of H.[2] This can be done by replacing every vertex v of degree d with $d > 4$ by a path of $d - 2$ vertices $v^1, \ldots, v^{d-2}$ of degree three.

- Construct a maximal spanning forest T for H with $\mathrm{er}(H, T) \leq 2r$ and $\mathrm{vr}(H, T) \leq 3r - 1$ using Proposition 5.2.7.

- According to Proposition 5.2.6, construct a tree decomposition for H with at most $2|V(H)|-1$ nodes and treewidth bounded by $\max(\mathrm{vr}(G, T), \mathrm{er}(G, T) + 1) \leq 3r' - 1 \leq 3r - 1$.

- Use Lemma 5.1.6 to get a tree decomposition for G. □

5.2.3 The Layerwise Separation Property (LSP)

We now develop a general scheme how to, given a planar graph problem with parameter k, construct tree decompositions of width $O(\sqrt{k})$. The idea is to combine the results of Proposition 5.2.3 and Theorem 5.2.4. An approach using these ingredients was first studied in our work [4]. However, the methods were very problem-specific and tailored towards the par-DOMINATING SET problem on planar graphs. In this subsection, we partly follow [8] where we broadened and methodologized the approach in [4] to make it applicable to other planar graph problems as well.

[2] This step is necessary, since Proposition 5.2.7 requires degree bounded by three.

5.2.3.1 Definition

In this section, we exploit the layer-structure of a plane graph in order to gain a "nice" separation of the graph. It is important that a "yes"-instance (G, k) (where G is a plane graph) of the parameterized graph problem under consideration admits a so-called "layerwise separation" of small size. By this, we mean, roughly speaking, a separation of the plane graph G (i.e., a collection of separators for G), such that each separator is contained in the union of constantly many subsequent layers (see conditions (i) and (ii) of the following definition). For (fixed-parameter) algorithmic purposes, it will be important that the corresponding separators are "small" (see condition (iii) of the definition).

Definition 5.2.8 *Let* $(G = (V, E), \phi)$ *be a plane graph of outerplanarity* $r := \mathrm{out}(G, \phi)$*, and let* $\mathcal{L}(G, \phi) = (L_1, \ldots, L_r)$ *be its layer decomposition. A* layerwise separation of width w and size s *of* (G, ϕ) *is a sequence* $(S_1, \ldots, S_r)$ *of subsets of* V*, with the properties that, for* $1 \le i \le r$:[3]

(i) $S_i \subseteq \bigcup_{j=i}^{i+(w-1)} L_j$,

(ii) S_i *separates layers* L_{i-1} *and* L_{i+w},

(iii) $\sum_{j=1}^{r} |S_j| \le s$.

Example 5.2.9 Trivially, setting $S_i := L_i$, every plane graph has a layerwise separation of width 1 and size n, where n is the number of vertices in the graph.

The crucial property that will guarantee the existence of fast tree decomposition based algorithms is what we call the "Layerwise Separation Property."

Definition 5.2.10 *A parameterized problem* par-$\mathcal{G}$ *on planar graphs is said to have the* Layerwise Separation Property *(abbreviated by: LSP) of width* w *and size-factor* d *if for each* $(G, k) \in$ par-$\mathcal{G}$ *and every planar embedding* ϕ *of* G*, the plane graph* (G, ϕ) *admits a layerwise separation of width* w *and size* dk.

In the following, we will give some examples of parameterized problems that have the LSP.

5.2.3.2 Linear Problem Kernels and the LSP

The LSP follows almost trivially for all graph problems which admit a linear problem kernel (see Chapter 2 for details).

Lemma 5.2.11 *Let* par-$\mathcal{G}$ *be a parameterized problem on planar graphs that admits a problem kernel of size* dk*. Then, the parameterized problem* par-$\mathcal{G}'$ *where each instance is replaced by its problem kernel has the LSP of width 1 and size-factor* d.

[3] By default, we let $S_i := \emptyset$ for all $i < 1$ and $i > r$.

Proof: Let $(G', k') \in \text{par-}\mathcal{G}'$ with $k' \leq dk$ be the problem kernel of $(G, k) \in \text{par-}\mathcal{G}$, and let $\mathcal{L}(G', \phi') = (L'_1, \dots, L'_{r'})$ be the layer decomposition of (G', ϕ') (where ϕ' is any embedding). Let $r' = \text{out}(G', \phi')$. Observe that $r' \leq \frac{dk}{3}$ since each layer has to consist of at least 3 vertices. Then, clearly, the sequence $S_i := L'_i$, $1 \leq i \leq r'$, is a layerwise separation of width 1 and size dk of (G', ϕ'). □

Example 5.2.12 (i) With the problem kernel of size $2k$ for par-VERTEX COVER (Corollary 2.1.8), we derive that par-VERTEX COVER on planar graphs has the LSP of width 1 and size-factor 2 on the set of reduced instances.

(ii) Using the $4k$ problem kernel for par-INDEPENDENT SET on planar graphs (see Proposition 2.1.4), we see that this problem has the LSP of width 1 and size-factor 4 on the set of reduced instances.

(iii) By Theorem 2.2.1, we obtain that par-DOMINATING SET on planar graphs has the LSP of width 1 and size-factor 335 on the set of reduced instances.

As already pointed out in Remark 2.2.2, we do not know whether there is a linear problem kernel for variants of the par-DOMINATING SET problem on planar graphs such as some of the par-DOMINATING SET WITH PROPERTY P problems presented in Subsection 1.3.4. However, in the next paragraph, we will establish the LSP for most of these variants. In this sense, the class of parameterized problems having the LSP is bigger than the one admitting linear problem kernels.

5.2.3.3 Directly Showing the LSP

It appears to us that directly showing that a parameterized graph problem has the LSP seems much easier than proving a linear problem kernel for the given problem.

Vertex Cover. As an example, consider par-VERTEX COVER on planar graphs, where we obtain the LSP in an almost trivial way without applying the heavy Nemhauser and Trotter machinery (see Corollary 2.1.8) as it is done in Example 5.2.12.(i).

Lemma 5.2.13 par-VERTEX COVER *on planar graphs has the LSP width* $w = 2$ *and size-factor* $d = 2$.

Proof: Let $(G, k) \in$ par-VERTEX COVER and let $\mathcal{L}(G, \phi) = (L_1, \dots, L_r)$ be the layer decomposition for some planar embedding ϕ of G. For a "witnessing" vertex cover V' of size k, the sets $S_i := (L_i \cup L_{i+1}) \cap V'$, $1 \leq i \leq r$, trivially form a layerwise separation. □

Dominating Set. From a historical point of view, at the point when we proved the LSP for par-DOMINATING SET on planar graphs in [4], we were not aware of the linear problem kernel for this problem. Showing a linear problem kernel (see Section 2.2) turned out to be much harder taking about two more years of research than directly showing the LSP. Besides, the

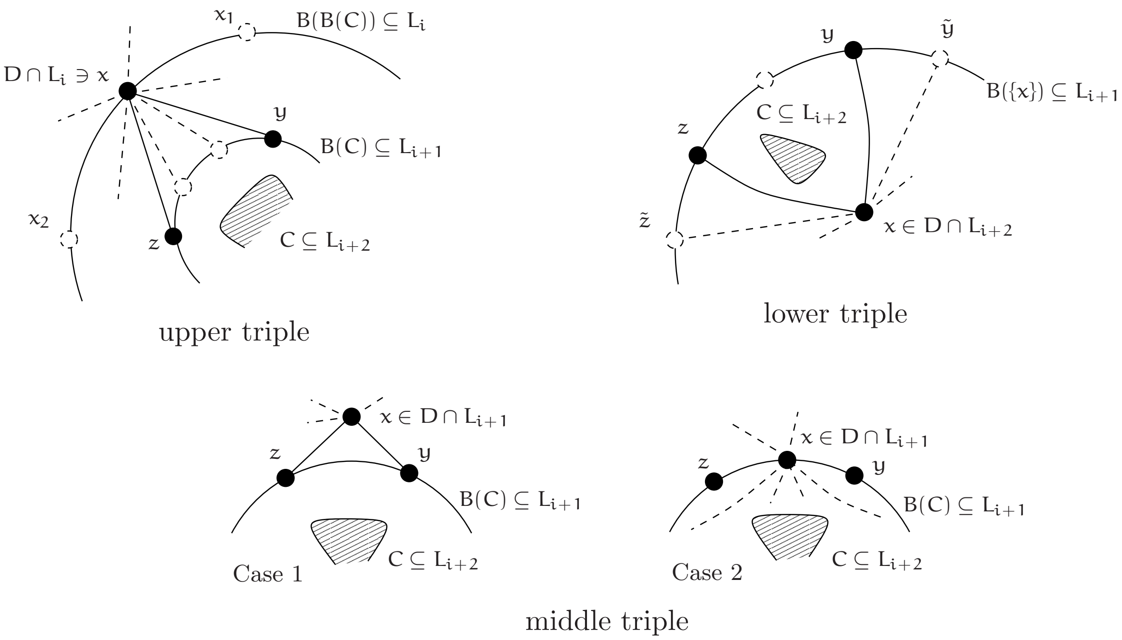

Figure 5.3: Construction of an upper triple associated with a vertex $x \in D \cap L_i$, a lower triple associated with a vertex $x \in D \cap L_{i+2}$ and a middle triple associated with a vertex $x \in D \cap L_i$. The union of all triples form the separator S_i.

constants which could be obtained from a direct proof of the LSP are even better than the ones in Example 5.2.12.(iii).

Proposition 5.2.14 par-DOMINATING SET *on planar graphs has the LSP width* $w = 3$ *and size-factor* $d = 51$.

We briefly sketch this direct proof here (for further details the reader is referred to [4]).[4]

Suppose $(G, k) \in$ par-DOMINATING SET and let $\mathcal{L}(G, \phi) = (L_1, \ldots, L_r)$ be the layer decomposition for some planar embedding ϕ of G. By Definitions 5.2.8 and 5.2.10, we need to construct sets $S_i \subseteq L_i \cup L_{i+1} \cup L_{i+2}$, $1 \leq i \leq r-2$) separating layer L_{i-1} from layer L_{i+3} in such a way that the total size of these sets can be bounded by some linear term in k. Suppose, we are given some "witnessing" dominating set D of size k. The key idea for proving that S_i separates layers L_{i-1} from L_{i+3} relies on a close investigation of the paths leaving layer L_{i-1} to the interior of the graph. Each such path passes a "first" vertex in layer L_{i+1}. This particular vertex can be dominated by vertices from $D \cap (L_i \cup L_{i+1} \cup L_{i+2})$. It turns out that, in order to cut this particular path, the set S_i has to contain the vertices of the sets $D \cap L_i$, $D \cap L_{i+1}$, and $D \cap L_{i+2}$ *plus* some suitably chosen pairs of neighbors of any of these vertices. This results in letting S_i be the

[4]Note that in the case of par-DOMINATING SET on planar graphs a similar construction as employed for par-VERTEX COVER (i.e., obtaining the separators S_i by intersecting a "witnessing" dominating set V' of G with a sequence of subsequent layers, e.g, $S_i := (L_{i-1} \cup L_i \cup L_{i+1}) \cap V'$), does not fulfill the conditions of a layerwise separation, since, in general, S_i need not be a separator.

union of so-called "upper," "lower," and "middle" triples. We will carry out the to some extent technical step in what follows. For the construction below, it is helpful to consider Fig. 5.3.

Let C be a component of layer L_{i+2} which is not a leaf in the layer decomposition tree (i.e., which contains vertices of layer L_{i+3} in its interior. Denote by $B(C)$ the unique shortest cycle in layer L_{i+1} that encloses C. We call $B(C)$ the *boundary cycle* for C. By $B(B(C))$ we denote the unique shortest cycle in layer L_i that encloses $B(C)$.

UPPER TRIPLES. Let $x \in D \cap L_i$ be a vertex that has a neighbor in $B(C)$ (see Fig. 5.3).[5] Let x_1 and x_2 be the neighbors of x on the boundary cycle $B(B(C))$. Starting from x_1, we go around x up to x_2 so that we visit all neighbors of x in layer L_{i+1}. We note the neighbors of x on the boundary cycle $B(C)$. Going around gives two outermost neighbors y and z on this boundary cycle. The *upper triple* for layer L_i associated with x, then, is the three-element set $\{x, y, z\}$.

MIDDLE TRIPLE. Let $x \in D \cap L_{i+1}$ be a vertex that has a neighbor in $B(C)$ (see Fig. 5.3). Note that, due to the layer model, it is easy to see that a vertex $x \in D \cap L_i$ can have at most two neighbors y, z in $B(C)$. Depending on whether x itself lies on the cycle $B(C)$ or not, we obtain two different cases which are both illustrated in Fig. 5.3. The *middle triple* for layer L_i associated with x, then, is the three-element set $\{x, y, z\}$.

LOWER TRIPLE. Let $x \in D \cap L_{i+2}$ be a vertex such that C is enclosed by the boundary cycle $B(\{x\})$ (see Fig. 5.3). For each pair $\tilde{y}, \tilde{z} \in B(\{x\}) \cap N(x)$ (where $\tilde{y} \neq \tilde{z}$), we consider the path $P_{\tilde{y},\tilde{z}}$ from $\tilde{y}$ to $\tilde{z}$ along the cycle $B(\{x\})$, taking the direction such that the region enclosed by $\{\tilde{z}, x\}$, $\{x, \tilde{y}\}$, and $P_{\tilde{y},\tilde{z}}$ contains the layer component C. Let $\{y, z\} \subseteq B(\{x\}) \cap N(x)$ be the pair such that the corresponding path $P_{y,z}$ is shortest. The *middle triple* for layer L_i associated with x, then, is the three-element set $\{x, y, z\}$.

Definition 5.2.15 *We define the set S_i as the union of all upper triples, lower triples and middle triples of layer L_i.*

With these definitions we claim the following statement which finishes the proof for Proposition 5.2.14.

Lemma 5.2.16 *The sets $(S_1, \ldots, S_r)$ as defined above yield a layerwise separation for G of width $w = 3$ and size-factor $d = 51$.*

Proof: The proof is deferred to the Appendix at the end of the chapter. □

Note that the size-factor given in Lemma 5.2.16 still leaves room for further improvements. By a more detailed investigation, our worst-case upper bound probably can be improved. New attempts in this direction are given in [97, 123] (also refer to Section 7.2).

The following corollary provides examples for parameterized graph problems which have the LSP, whereas a linear problem kernel is unknown (see Remark 2.2.2).

Corollary 5.2.17 *The* par-DOMINATING SET WITH PROPERTY P *problem on planar graphs has the LSP of width* 3 *and size-factor* 51.

[5]Then, clearly, $x \in B(B(C))$, by definition of a boundary cycle.

Proof: Since every dominating set with property P is, in particular, a dominating set, the above construction of a layerwise separation can be carried over. $\square$

5.2.4 New Constructive Upper Bounds for the Treewidth

In this subsection, we summarize the partial results from the previous paragraphs to provide new upper bounds for the treewidth $\mathrm{tw}(G)$ of a planar graph G. The upper bounds will be given in terms of various graph problem parameters, such as the vertex cover number $\mathrm{vc}(G)$, the domination number $\mathrm{ds}(G)$, or the stability number $\mathrm{is}(G)$ (see Definitions 1.3.3, 1.3.4, and 1.3.5). Moreover, we will see that the bounds derived here are asymptotically optimal and that corresponding tree decompositions can be constructed very efficiently. Our approach will work for all problems that have the LSP.

The following remark gives a (trivial) relation of the treewidth to some problem parameters.

Remark 5.2.18 Let G be a planar graph with layer decomposition $\mathcal{L}(G,\phi) = (L_1,\dots,L_r)$ for some embedding ϕ such that $\mathrm{out}(G) = \mathrm{out}((G,\phi))$. Firstly, observe that, by definition of a layer, for each $1 \le i \le r-1$, $G[L_i]$ contains a cycle C_3 of length three as a minor.[6] Since we need at least two vertices to cover the edges of a C_3, we get $\mathrm{out}(G) \le \frac{1}{2}\mathrm{vc}(G)+1$. And secondly, since each vertex in a dominating set can dominate vertices from the previous, the next, or its own layer only, we obtain $\mathrm{out}(G) \le 3\,\mathrm{ds}(G)$. Using the result of Subsection 5.2.2, this implies the following trivial upper linear bounds for the treewidth:[7]

$$\begin{aligned}\mathrm{tw}(G) &\le \frac{3}{2}\mathrm{vc}(G)+2, \quad \text{and}\\ \mathrm{tw}(G) &\le 9\,\mathrm{ds}(G)-3.\end{aligned}$$

Note that for general graphs, no relation of the form $\mathrm{tw}(G) \le f(\mathrm{ds}(G))$ (for any function f) holds; consider, e.g., the clique K_n with n vertices, where $\mathrm{tw}(K_n) = n-1$, but $\mathrm{ds}(K_n) = 1$.

Our goal, however, is to derive *sublinear* upper bounds for planar graphs. Alon *et al.* [20] gave a first result in this direction.

Proposition 5.2.19 *Let G be a graph which excludes K_h as a minor, h being some integer, then $\mathrm{tw}(G) \le h^{\frac{3}{2}}\sqrt{n}$, where n is the number of vertices of G.*

In particular, for planar graphs (which exclude a K_5 as a minor), we get

$$\mathrm{tw}(G) \le 5^{\frac{3}{2}}\sqrt{n} \approx 11.18\sqrt{n}. \tag{5.1}$$

Unfortunately, the proof for the above relation is non-constructive. Moreover, since we are interested in designing fixed-parameter algorithms, a major drawback of the above relation is that this is *not* an upper bound in terms of the problem parameters.

[6] Layer L_r may consist of a single vertex only.

[7] It is even true that $\mathrm{tw}(G) \le \mathrm{vc}(G)$ holds for general graphs. This estimate is sharp (which becomes clear by, again, considering the graph K_n, where $\mathrm{vc}(K_n) = n-1$).

5.2.4.1 Partial Layerwise Separation

The idea of the following is that, from a layerwise separation of small size (say bounded by $O(k)$), we are able to choose a set of separators such that their size is bounded by $O(\sqrt{k})$ and—at the same time—the subgraphs into which these separators cut the original graph have outerplanarity bounded by $O(\sqrt{k})$.

In order to formalize such a choice of separators from a layerwise separation, we give the following fairly technical definition.

Definition 5.2.20 *Let* $(G = (V, E), \phi)$ *be a plane graph with layer decomposition* $\mathcal{L}(G, \phi) = (L_1, \ldots, L_r)$. *A* partial layerwise separation of width w *is a sequence* $\mathcal{S} = (S_1, \ldots, S_q)$ *such that there exist* $1 = i_0 < i_1 < \ldots < i_q < i_{q+1} = r$ *such that for* $1 \leq i \leq q$:[8]

(i) $S_j \subseteq \bigcup_{\ell=i_j}^{i_j+(w-1)} L_\ell$,

(ii) $i_j + w \leq i_{j+1}$ *(so, the sets in* $\mathcal{S}$ *are pairwise disjoint) and*

(iii) S_j *separates layers* L_{i_j-1} *and* L_{i_j+w}.

The sequence $\mathcal{C}_{\mathcal{S}} = (G_0, \ldots, G_q)$ *with*

$$G_j := G[(\bigcup_{\ell=i_j}^{i_{j+1}+(w-1)} L_\ell) - (S_j \cup S_{j+1})], \quad j = 0, \ldots, q,$$

is called the sequence of graph chunks *obtained by* $\mathcal{S}$.

With this definition at hand, we can state the key result needed to establish the new upper bounds for the treewidth of plane graphs that admit a layerwise separation.

Proposition 5.2.21 *Let* $(G = (V, E), \phi)$ *be a plane graph with* n *vertices that admits a layerwise separation of width* w *and size* s. *Then, for every* $\psi \in \mathbb{R}_+$, *there exists a partial layerwise separation* $\mathcal{S}(\psi)$ *of width* w *such that*

(i) $\max_{S \in \mathcal{S}(\psi)} |S| \leq \psi\sqrt{s}$ *and*

(ii) $\mathrm{out}(H) \leq (\sqrt{s}/\psi) + w$ *for each graph chunk* H *in* $\mathcal{C}_{\mathcal{S}(\psi)}$.

Moreover, there is an algorithm with running time $O(\sqrt{s}\, n)$ *which, for a given* ψ,

- *recognizes whether* (G, ϕ) *admits a layerwise separation of width* w *and size* s *and, if so,*
- *computes a partial layerwise separation* $\mathcal{S}(\psi)$ *of width* w *that fulfills the conditions above.*

[8] Again, by default, we set $S_i := \emptyset$ for $i < 1$ and $i > q$.

```
procedure partial_layerwise_separation
/* input:   a plane graph (G, φ) which admits a layerwise separation of width w and size s.  */
/*          and a "trade-off parameter ψ.                                                    */
/* output:  a partial layerwise separation of width w                                       */
/*          that fulfills properties (i)+(ii) from Proposition 5.2.21.                      */
```

- let $s_0 := \psi\sqrt{s}$, and let $\mathcal{S}(\psi) =$ `emptylist`
- `for` $j = 1, \ldots, r - w$ `do`
  - let $G_j(v_1, v_2)$ denote the graph $G[\bigcup_{\ell=j}^{j+(w-1)} L_\ell]$ with two further vertices v_1 and v_2 and edges from v_1 to all vertices in L_j and from v_2 to all vertices in L_{j+w-1}.
  - `if` ($G_j(v_1, v_2)$ admits a v_1-v_2 separator S of size at most s_0) `then`
    - $\mathcal{S}(\psi)$.`append`(S)
    - $j \leftarrow j + w$
- suppose $\mathcal{S}(\psi) = (S_1, \ldots, S_q)$ is the sequence of all separators of size at most s_0.
- `if` $(\exists j_0 : i_{j_0+1} - i_{j_0} > \frac{\sqrt{s}}{\psi})$ `then`
  - `return` "(G, ϕ) admits no layerwise separation of width w and size s"; `else`
  - `return` $\mathcal{S}(\psi)$

Figure 5.4: Construction of a partial layerwise separation according to Proposition 5.2.21.

Proof: For $1 \leq m \leq w$, consider the integer sequences $I_m = (m + jw)_{j=0}^{\lfloor r/w \rfloor - 1}$ and the corresponding sequences of separators $\mathcal{S}_m = (S_i)_{i \in I_m}$. Note that each $\mathcal{S}_m$ is a sequence of pairwise disjoint separators. Since $(S_1, \ldots, S_r)$ is a layerwise separation of size s, this implies that there exists a $1 \leq m' \leq w$ with

$$\sum_{i \in I_{m'}} |S_i| \leq \frac{s}{w}. \tag{5.2}$$

For a given ψ, let $s_0 := \psi\sqrt{s}$. Define $\mathcal{S}(\psi)$ to be the subsequence of $\mathcal{S}_{m'}$ such that $|S| \leq s_0$ for all $S \in \mathcal{S}(\psi)$, and $|S| > s_0$ for all $S \in \mathcal{S}_{m'} - \mathcal{S}(\psi)$. This yields condition (i). As to condition (ii), suppose that $\mathcal{S}(\psi) = (S_{i_1}, \ldots, S_{i_q})$. How many layers are two separators S_{i_j} and $S_{i_{j+1}}$ apart? To answer this, note that the number of separators in $\mathcal{S}_{m'}$ that appear between S_{i_j} and $S_{i_{j+1}}$ is $(i_{j+1} - i_j)/w$. Since all of these separators have size greater than s_0, their number has to be bounded by $s/(w\, s_0)$, see Eq. (5.2). Therefore, we get $i_{j+1} - i_j \leq \sqrt{s}/\psi$ for all $j = 1, \ldots, q - 1$. Hence, the chunks $G[(\bigcup_{\ell=i_j}^{i_{j+1}+w-1} L_\ell) - (S_{i_j} \cup S_{i_{j+1}})]$ have outerplanarity at most $\sqrt{s}/\psi + w$.

The algorithm that computes a partial layerwise separation $\mathcal{S}$ is depicted in Fig. 5.4. The correctness of this algorithm is established by the following arguments. Firstly, note that a v_1-v_2 separator S of the graph $G_j(v_1, v_2)$ (as defined in the algorithm in Fig. 5.4), by construction, is a separator of G that lies in the w many consecutive layers $L_j \cup \cdots \cup L_{j+(w-1)}$ and separates layers L_{j-1} and L_{j+w}. In this sense, all separators found by the algorithm in Fig. 5.4 have properties (i) and (iii) of Definition 5.2.8. Whenever a new separator is found, by letting $j \leftarrow j + w$, we make sure that the set $\mathcal{S}(\psi)$ also has property (ii) of Definition 5.2.8, i.e., $\mathcal{S}(\psi)$ indeed

```
procedure tree_decomposition
```

/* **input:** a plane graph (G, ϕ) which admits a layerwise separation of width w and size s */
/* and a trade-off parameter ψ. */
/* **output:** a tree decomposition $\mathcal{X}_\psi$ with $\mathrm{tw}(\mathcal{X}_\psi) \leq (2\psi + 3/\psi)\sqrt{s} + (3w-1)$. */

- compute a partial layerwise separation $\mathcal{S}(\psi) = (S_1, \ldots, S_q)$ of width w with corresponding graph chunks $\mathcal{C}_{\mathcal{S}(\psi)} = (G_0, \ldots, G_q)$, such that

$$\max_{S \in \mathcal{S}(\psi)} |S| \;\leq\; \psi\sqrt{s}, \quad \text{and} \quad \mathrm{out}(G_i) \;\leq\; \frac{\sqrt{s}}{\psi} + w$$

  for all $i = 0, \ldots, q$ (using the algorithm from Proposition 5.2.21).

- construct a tree decomposition $\mathcal{X}_i$ of width at most $3\,\mathrm{out}(G_i) - 1$ for each of the graphs G_i (using the algorithm from Theorem 5.2.4).

- join the tree decompositions $\mathcal{X}_\psi^{(0)}, \ldots, \mathcal{X}_\psi^{(q)}$ via the separators $S_1, \ldots, S_q$ in order to obtain a global tree decomposition $\mathcal{X}_\psi$ for G (using the merging technique explained in Proposition 5.2.3).

Figure 5.5: Construction of a tree decomposition according to Theorem 5.2.23.

is a partial layerwise separation. The fact that $\max_{S \in \mathcal{S}(\psi)} |S| \leq s_0 = \psi\sqrt{s}$ is trivial to see.

Secondly, by the arguments given above, no two separators S_{j_0} and S_{j_0+1} of size at most s_0 can be more than $\sqrt{s}/\psi$ layers apart. Hence, if there was a j_0 such that $i_{j_0+1} - i_{j_0} > \sqrt{s}/\psi$, the algorithms correctly exits and returns "no." Otherwise, we obtain $\mathrm{out}(H) \leq (\sqrt{s}/\psi) + w$ for each graph chunk H in $\mathcal{C}_{\mathcal{S}(\psi)}$.

Concerning the running time of this algorithm, we remark that a the separator S_j, for $1 \leq j \leq r - w$, of size at most s_0 in the graphs $G_j(v_1, v_2)$ can be computed in time $O(s_0 n)$ using techniques based on maximum flow (see [122] for details). □

Remark 5.2.22 In what follows, the positive real number ψ of Proposition 5.2.21 is also called *trade-off parameter*. This is because it allows us to optimize the trade-off between outerplanarity and separator size.

5.2.4.2 LSP and Treewidth

Proposition 5.2.21 will help to construct a tree decomposition of treewidth $\mathrm{tw}(G) = O(\sqrt{k})$, assuming that a layerwise separation of some constant width and size dk exists.

Theorem 5.2.23 *For a plane graph (G, ϕ) with n vertices that admits a layerwise separation of width w and size s, we have*

$$\mathrm{tw}(G) \;\leq\; 2\sqrt{6s} + (3w - 1).$$

A corresponding tree decomposition can be computed in time $O(\sqrt{s}\,n)$.

Proof: Let $\psi \in \mathbb{R}_+$ be some trade-off parameter (the optimal value to be determined later). An algorithm which constructs a tree decomposition $\mathfrak{X}_\psi$ is given in Fig. 5.5. We now give an upper bound on the width of the resulting tree decomposition $\mathcal{X}_\psi$. By Proposition 5.2.3, we obtain

$$\begin{aligned} \mathrm{tw}(\mathcal{X}_\psi) &\leq 2 \max_{S \in \mathcal{S}(\psi)} |S| + \max_{i=0,\ldots,q} \mathrm{tw}(G_i) \\ &\leq 2 \max_{S \in \mathcal{S}(\psi)} |S| + 3(\max_{i=0,\ldots,q} \mathrm{out}(G_i)) - 1 \\ &\leq (2\psi + 3/\psi)\sqrt{s} + (3w - 1). \end{aligned}$$

This upper bound is minimized for $\psi = \sqrt{3/2}$. Therefore, $\mathrm{tw}(\mathcal{X}_\psi) \leq 2\sqrt{6s} + (3w-1)$. The most cost-expensive step is to construct the tree decompositions $\mathcal{X}_\psi^{(i)}$, which can be done in time $O(\sum_{i=0}^{q} \mathrm{out}(G_i) \cdot |V(G_i)|) = O(\sqrt{s}n)$. □

Plugging in Example 5.2.9, we obtain the following corollary which outperforms the result of Alon, Seymour, and Thomas (see Proposition 5.2.19 and Eq. (5.1)) for the planar case.

Corollary 5.2.24 *For a planar graph* G *with* n *vertices, we have*

$$\mathrm{tw}(G) \leq 2\sqrt{6n} + 2 \approx 4.90\sqrt{n} + 2.$$

The following is our main result. It will be used to design fixed-parameter algorithms with running time $O(2^{O(\sqrt{k})} n)$ for graph problems which have the Layerwise Separation Property.

Corollary 5.2.25 *Let* par-$\mathcal{G}$ *be a parameterized problem on planar graphs and assume that* par-$\mathcal{G}$ *has the LSP of width* w *and size-factor* d*. Then there is an algorithm that outputs, for an instance* (G,k)*, in time* $O(\sqrt{k}|V(G)|)$ *either that* $(G,k) \notin$ par-$\mathcal{G}$*, or computes a tree decomposition* $\mathcal{X}$ *for* G *of width bounded by*

$$\mathrm{tw}(\mathcal{X}) \leq 2\sqrt{6dk} + (3w-1).$$

Proof: This is a consequence of Definition 5.2.10, Proposition 5.2.21, and Theorem 5.2.23. □

In particular, we can derive upper bounds of the treewidth of a planar graph in terms of several graph specific numbers, which clearly improve the trivial bounds in Remark 5.2.18.

Corollary 5.2.26 *Let* G *be a planar graph with* n *vertices.*

(i) *We have*

$$\mathrm{tw}(G) \leq 4\sqrt{3\,\mathrm{vc}(G)} + 5 \approx 6.93\sqrt{\mathrm{vc}(G)} + 5, \tag{5.3}$$

and a corresponding tree decomposition can be constructed in time $O(\sqrt{\mathrm{vc}(G)}\,n)$.

(ii) *We have*

$$\mathrm{tw}(G) \leq 6\sqrt{34\,\mathrm{ds}(G)} + 8 \approx 34.99\sqrt{\mathrm{ds}(G)} + 8, \tag{5.4}$$

and a corresponding tree decomposition can be constructed in time $O(\sqrt{\mathrm{ds}(G)}\,n)$.

The upper bounds are both asymptotically optimal in the sense that there exist graphs G_n *with* n *vertices (*$n \in \mathbb{N}$*) such that* $\mathrm{tw}(G_n) = \Omega(\sqrt{\mathrm{vc}(G_n)})$ *and* $\mathrm{tw}(G_n) = \Omega(\sqrt{\mathrm{ds}(G_n)})$.

Proof: The result follows from Corollary 5.2.25 when plugging in Lemma 5.2.13 (for par-VERTEX COVER) and Proposition 5.2.14 (for par-DOMINATING SET).

For the lower bounds, we use the grid graphs $G_n = P_m \times P_m$ with $n = m^2$ vertices, where $tw(G_n) = m$ (see Corollary 5.1.5). Observe that for the grid graph, we have $\mathrm{vc}(G_n) = \lfloor m^2/2 \rfloor$ (which is easy to see) and that $\mathrm{ds}(G_n) \leq \frac{1}{5}(m^2 + m - 3)$ (see [63]). □

5.3 Dynamic Programming on Tree Decompositions

We now turn our attention to "Phase 2" of the general scheme of a tree decomposition based algorithm as introduced at the beginning of this chapter. It was already mentioned there that many NP-complete graph problems turn out to be solvable in polynomial or even linear time when restricted to the class $\mathbb{G}_{\mathrm{tw}\leq\ell}$ of graphs of treewidth bounded by ℓ. The corresponding algorithms typically rely on a dynamic programming strategy [41]. A similar technique that can be characterized by "computing tables of characterizations of partial solutions" appeared in [25]. An overview over these and other methods—like graph reduction [24] and monadic second order logic [67]—to obtain polynomial time algorithms for graph problems on $\mathbb{G}_{\mathrm{tw}\leq\ell}$ can be found in [44]. Here, we present new dynamic programming algorithms for domination-like problems that significantly improve previous work [66, 183, 184].

5.3.1 The Basic Concept

In this short subsection, we sketch a general tree decomposition based dynamic programming scheme. Moreover, we summarize previous results and relate them to our improved algorithms.

5.3.1.1 Tree Decomposition Based Dynamic Programming

The algorithm in Fig. 5.6 informally describes the general structure of a dynamic programming strategy on a graph $G = (V, E)$ with a given tree decomposition of width bounded by ℓ in order to solve some graph problem.

As an example take the VERTEX COVER problem. Here, in each table A_i, we would have $2^{|X_i|}$ entries for all possible configurations of whether a vertex in X_i belongs to the vertex cover or not. For each configuration, one would store how many vertices are needed for a vertex cover using exactly this configuration. The most cost-expensive part of the algorithm typically is the updating process for a JOIN NODE. In the case of VERTEX COVER, a brute force algorithm would, for a JOIN NODE i with children i_1 and i_2, compare each entry of A_{i_1} with each entry of A_{i_2} in order to update A_i, resulting in $O(2^{2\ell}) = O(4^\ell)$ comparisons per JOIN NODE. By a more careful comparison procedure this can be brought down to only $O(2^\ell)$ comparisons per JOIN NODE (see [14, 15]). We do not go into detail for the (relatively easy) VERTEX COVER problem but, in the following, describe in a more formal way how to deal with the more involved DOMINATING SET problem.

- Transform the tree decomposition into a nice tree decomposition $\mathcal{X} = \langle \{X_i \mid i \in V(T)\}, T \rangle$ (see Lemma 5.1.3).
- For each node $i \in V(T)$, we keep a table A_i, which stores all necessary information on possible solutions of the graph problem restricted to $G[X_i]$.
- The dynamic programming is performed in three steps in a bottom-up order of the tree T.

  Step 1 (initialization): All tables A_i for the LEAF NODES i are initialized.

  Step 2 (updating process): A table A_i for an inner node i is computed using the graph $G[X_i]$ and the information of the tables corresponding to the child(ren) of i. The updating process depends on the type of the node i (i.e., FORGET NODE, INTRODUCE NODE or JOIN NODE).

  Step 3 (final evaluation): The solution of the problem can be found by inspecting the table A_r of the root node r of T.

Figure 5.6: General scheme for a dynamic programming on a given tree decompositions.

5.3.1.2 Previous Work and Overview of our Results

Most of the previous work on this subject concentrated on showing that a certain problem *can* be solved in polynomial or even linear time on $\mathbb{G}_{\mathrm{tw}\leq\ell}$, the running time typically being some exponential term in ℓ. So far, only little attention has been paid to a quantitative study of this exponential growth. Since our concern is to derive efficient fixed-parameter algorithms, a central question is to get the combinatorial explosion in ℓ as small as possible—this is what we investigate here.

To our knowledge, the best previous results for (dynamic programming) algorithms on tree decompositions applied to "domination-like" problems were obtained by Telle and Proskurowski [183, 184]. For a graph with a tree decomposition of N tree nodes and width ℓ, Telle and Proskurowski solved DOMINATING SET in time $O(9^\ell N)$. Ten years earlier, Corneil and Keil presented a time $O(4^\ell N^{\ell+2})$ algorithm for ℓ-trees [66], an equivalent definition of the class $\mathbb{G}_{\mathrm{tw}\leq\ell}$. Observe, however, that the latter algorithm is *not* a fixed-parameter algorithm. Moreover, Telle and Proskurowski [183, 184] provide a general classification of "domination-like" problems (which we refer to as DOMINATING SET WITH PROPERTY P) and give algorithms of running time $O((\sigma')^\ell N)$, where the base σ' depends on the graph property P. The last column of Table 5.1 summarizes the results from [183, 184]. In this section, we will introduce the new, in a sense general concept of "monotonicity" for dynamic programming for domination-like problems. Using this concept, we will give a time $O(4^\ell N)$ algorithm for DOMINATING SET on $\mathbb{G}_{\mathrm{tw}\leq\ell}$, a significant reduction of the combinatorial explosion.[9] Moreover, we improve basically all running times for domination-like problems given by Telle and Proskurowski (our improved bases σ over the bases σ' from [183, 184] are summarized in Table 5.1). For example, we can solve TOTAL DOMINATING SET problem in time $O(6^\ell N)$, where Telle and Proskurowski had $O(16^\ell N)$.

[9] Observe that in the conference version of [4] we gave wrong bounds for the dynamic programming algorithm for DOMINATING SET, claiming a running time $O(3^\ell N)$. This was due to a misinterpretation of [184, Theorem 5.7] where the correct base of the exponential term in the running time is $3^2 = 9$ instead of 3.

Problem	λ	σ	σ'
(WEIGHTED) DOMINATING SET	3	4	9
INDEPENDENT DOMINATING SET	3	4	9
PERFECT DOMINATING SET	3	4	9
PERFECT CODE	3	4	9
TOTAL DOMINATING SET	4	6	16
TOTAL PERFECT DOMINATING SET	4	6	16
RED-BLUE DOMINATING SET	4	4	-
VERTEX COVER	2	2	4
INDEPENDENT SET	2	2	4

Table 5.1: **Summary of our results** (Theorems 5.3.1, 5.3.3, 5.3.5) **and comparison with previous work**. The entries in the third column give the base σ for our time $O(\sigma^{\ell} N)$ algorithm (ℓ being the width of the given tree decomposition and N being the number of nodes in the tree decomposition). The entries of the fourth column show the corresponding base values σ' of the so far best known algorithms by Telle and Proskurowski (see [183, Theorem 4, Table 1] and [184, Theorem 5.7]). The second column gives the number λ of colors used in our dynamic programming step.

To illustrate the significance of our results, in Table 5.2, we compare (hypothetical) running times of the $O(9^{\ell}N)$ algorithm of Telle and Proskurowski [183, 184] to our $O(4^{\ell}N)$ algorithm.

5.3.2 An Improved Algorithm for DOMINATING SET Based on Monotonicity

In this subsection, we present our main algorithm which is based on a fresh view on dynamic programming: Compared to previous work, we perform the updating process of the tables in a more careful, less time consuming way by making use of the "monotonous structure of the tables." We partly follow [14].

Theorem 5.3.1 DOMINATING SET *can be solved in time* $O(4^{\ell} N)$ *if an input graph* G *is given together with a tree decomposition of width* ℓ *and* N *tree nodes.*

The outline of the corresponding algorithm and its proof of correctness fill the rest of this subsection. From now on suppose that the given tree decomposition of our input graph $G = (V, E)$ is $\mathcal{X} = \langle \{X_i \mid i \in V(T)\}, T \rangle$. By Lemma 5.1.3, we can assume that $\mathcal{X}$ is a nice tree decomposition.

5.3.2.1 Colorings and Monotonicity

Suppose that $V = \{x_1, \ldots, x_n\}$. We assume that the vertices in the bags are ordered by their indices, i.e., $X_i = (x_{i_1}, \ldots, x_{i_{n_i}})$ with $i_1 \leq \ldots \leq i_{n_i}$ for all $i \in V(T)$.

Colorings. In the following, we use three different "colors" that will be assigned to the vertices in a bag:

Algorithm	$\ell = 5$	$\ell = 10$	$\ell = 15$	$\ell = 20$
$9^\ell N$	0.05 sec	1 hour	6.5 years	$3.9 \cdot 10^5$ years
$4^\ell N$	0.001 sec	1 sec	18 minutes	13 days

Table 5.2: Comparing our time $O(4^\ell N)$ algorithm for DOMINATING SET with the $O(9^\ell N)$ algorithm of Telle and Proskurowski in the case $N = 1000$ (number of nodes of the tree decomposition), we assume a machine executing 10^9 instructions per second and we neglect the constants hidden in the O-terms (which may not be in practice, but which are comparable in both cases).

- "black" (represented by 1, meaning that the vertex belongs to the dominating set),
- "white" (represented by 0, meaning that the vertex is already dominated at the current stage of the algorithm), and
- "grey" (represented by $\hat{0}$, meaning that, at the current stage of the algorithm, we still ask for a domination of this vertex).

A vector $c = (c_1, \dots, c_{n_i}) \in \{0, \hat{0}, 1\}^{n_i}$ will be called a *coloring* for the bag $X_i = (x_{i_1}, \dots, x_{i_{n_i}})$, and the *color* assigned to vertex x_{i_t} by the coloring c is given by the coordinate c_t.

According to the general scheme in Fig. 5.6, for each bag X_i (with $|X_i| = n_i$), we will use a mapping

$$A_i : \{0, \hat{0}, 1\}^{n_i} \longrightarrow \mathbb{N} \cup \{+\infty\}.$$

The semantics of the table entries is as follows: For a coloring $c = (c_1, \dots, c_{n_i}) \in \{0, \hat{0}, 1\}^{n_i}$, the value $A_i(c)$ stores how many vertices are needed for a minimum dominating set (of the graph visited up to the current stage of the algorithm) under the restriction that the color assigned to vertex x_{i_t} is c_t $(t = 1, \dots, n_i)$.

A coloring $c \in \{0, \hat{0}, 1\}^{n_i}$ is *locally invalid* for a bag X_i if

$$\big(\exists s \in \{1, \dots, n_i\} : c_s = 0\big) \wedge \big(\nexists t \in \{1, \dots, n_i\} : (x_{i_t} \in N(x_{i_s}) \wedge c_t = 1)\big).$$

In other words, a coloring is locally invalid if there is some vertex x_{i_s} in the bag that is colored white, but this color is not "justified" within the bag, i.e., x_{i_s} is not dominated by a vertex within the bag using this coloring.[10] Also, for a coloring $c = (c_1, \dots, c_m) \in \{0, \hat{0}, 1\}^m$ and a color $d \in \{0, \hat{0}, 1\}$, we use the notation $\#_d(c) := |\{t \in \{1, \dots, m\} : c_t = d\}|$.

Monotonicity. On the color set $\{0, \hat{0}, 1\}$, let $\prec$ be the partial ordering given by $\hat{0} \prec 0$ and $d \prec d$ for all $d \in \{0, \hat{0}, 1\}$. This ordering naturally extends to colorings: For $c = (c_1, \dots, c_m), c' = (c'_1, \dots, c'_m) \in \{0, \hat{0}, 1\}^m$, we let $c \prec c'$ iff $c_t \prec c'_t$ for all $1 \le t \le m$.

We call a mapping

$$A_i : \{0, \hat{0}, 1\}^{n_i} \longrightarrow \mathbb{N} \cup \{+\infty\}$$

[10] A locally invalid coloring still may be a correct coloring if the white vertex whose color is not "justified" *within* the bag already is dominated by a vertex from other bags.

monotonous from the partially ordered set $(\{0,\hat{0},1\}^{n_i},\prec)$ to $(\mathbb{N}\cup\{+\infty\},\leq)$ if for $c,c'\in\{0,\hat{0},1\}^{n_i}$, $c\prec c'$ implies $A(c)\leq A(c')$. It is very essential for the correctness of our algorithm as well as for the claimed running time that all the mappings A_i will be monotonous.

5.3.2.2 The Algorithm

We use the mappings introduced above to perform the dynamic programming approach as described in Fig. 5.6 in Subsection 5.3.1.1. Note that at each stage of the algorithm we take care that the mappings visited up to that stage are monotonous.

Step 1 (initialization). In the first step of the algorithm, for each LEAF NODE i of the tree decomposition, we initialize the mapping A_i:

`for all` $c\in\{0,\hat{0},1\}^{n_i}$ `do`

$$A_i(c) \leftarrow \begin{cases} +\infty & \text{if } c \text{ is locally invalid for } X_i \\ \#_1(c) & \text{otherwise} \end{cases} \tag{5.5}$$

By this initialization step, we make sure that only colorings are taken into consideration where an assignment of color 0 is justified.

Step 2 (updating process). After the initialization, we visit the bags of our tree decomposition bottom-up from the leaves to the root, evaluating the corresponding mappings in each step according to the following rules.

FORGET NODES: Suppose i is a FORGET NODE with child j and suppose that $X_i=(x_{i_1},\ldots,x_{i_{n_i}})$. W.l.o.g.[11], we may assume that $X_j=(x_{i_1},\ldots,x_{i_{n_i}},x)$. Evaluate the mapping A_i of X_i as follows:

`for all` $c\in\{0,\hat{0},1\}^{n_i}$ `do`

$$A_i(c) \leftarrow \min_{d\in\{0,1\}} A_j(c\times\{d\}) \tag{5.6}$$

Note that a coloring $c\times\{\hat{0}\}$ for X_j means that the vertex x is assigned color $\hat{0}$, i.e., not yet dominated up to this point of the algorithm. Since, by condition (iii) of Definition 5.1.1, the vertex x will never appear in a bag for the rest of the algorithm, a coloring $c\times\{\hat{0}\}$ will remain unresolved and it will not lead to a dominating set. That is why the minimum in the assignment (5.6) is taken over colors 1 and 0 only.

INTRODUCE NODES: Suppose that i is an INTRODUCE NODE with child j and suppose, furthermore, that $X_j=(x_{j_1},\ldots,x_{j_{n_j}})$. W.l.o.g.[12], we may assume that $X_i=(x_{j_1},\ldots,x_{j_{n_j}},x)$. Let $N(x)\cap X_i=\{x_{j_{p_1}},\ldots,x_{j_{p_s}}\}$ be the neighbors of the "introduced" vertex x that appear in

[11] Possibly after rearranging the vertices in X_j and the entries of A_j accordingly.
[12] Possibly after rearranging the vertices in X_i and the entries of A_i accordingly.

the bag X_i. We now define a function $f : \{0, \hat{0}, 1\}^{n_j} \to \{0, \hat{0}, 1\}^{n_j}$ on the set of colorings of X_j. For $c = (c_1, \ldots, c_{n_j}) \in \{0, \hat{0}, 1\}^{n_j}$, let $f(c) := (c'_1, \ldots, c'_{n_j})$ such that

$$c'_t = \begin{cases} \hat{0} & \text{if } t \in \{p_1, \ldots, p_s\} \text{ and } c_t = 0, \\ c_t & \text{otherwise.} \end{cases}$$

Then, evaluate the mapping A_i of X_i as follows:

`for all` $c = (c_1, \ldots, c_{n_j}) \in \{0, \hat{0}, 1\}^{n_j}$ `do`

$$A_i(c \times \{0\}) \leftarrow \begin{cases} A_j(c) & \text{if } x \text{ has a neighbor } x_{j_q} \text{ in } X_i \text{ with } c_q = 1, \\ +\infty & \text{otherwise} \end{cases} \tag{5.7}$$

$$A_i(c \times \{1\}) \leftarrow A_j(f(c)) + 1 \tag{5.8}$$

$$A_i(c \times \{\hat{0}\}) \leftarrow A_j(c) \tag{5.9}$$

For the correctness of the Assignments (5.7) and (5.8), we remark the following: It is clear that, if we assign color 0 to vertex x (see Assignment (5.7)), we again (as already done in the initializing step of Assignment (5.5)) have to check whether this color can be justified at the current stage of the algorithm. Such a justification is given if and only if the coloring under examination already assigns a 1 to some neighbor of x in X_i. This is true, since condition (iii) of Definition 5.1.1 implies that no neighbor of x has been considered in previous bags, and, hence, up to the current stage of the algorithm, x can only be dominated by a vertex in X_i (as checked in Assignment (5.7)).

If we assign color 1 to vertex x (see Assignment (5.8)), we already dominate all vertices $\{x_{j_{p_1}}, \ldots, x_{j_{p_s}}\}$. Suppose now we want to evaluate $A_i(c \times \{1\})$ and suppose some of these vertices are assigned color 0 by c, say $c_{p'_1} = \ldots = c_{p'_q} = 0$ (where $(p'_1, \ldots, p'_q)$ is a subsequence of $(p_1, \ldots, p_s)$). Since the "1-assignment" of x already justifies the "0-values" of $c_{p'_1}, \ldots, c_{p'_q}$, and since our mapping A_j is monotonous, we obtain $A_i(c \times \{1\})$ by taking entry $A_j(c')$, where $c'_{p'_1} = \ldots = c'_{p'_q} = \hat{0}$, i.e., where $c' = f(c)$.

JOIN NODES: Suppose i is a JOIN NODE with children j and k and suppose that $X_i = X_j = X_k = (x_{i_1}, \ldots, x_{i_{n_i}})$. Let $c = (c_1, \ldots, c_{n_i}) \in \{0, \hat{0}, 1\}^{n_i}$ be a coloring for X_i. We say that $c' = (c'_1, \ldots, c'_{n_i})$, $c'' = (c''_1, \ldots, c''_{n_i}) \in \{0, \hat{0}, 1\}^{n_i}$ *divide* c if

(i) $(c_t \in \{1, \hat{0}\} \Rightarrow c'_t = c''_t = c_t)$, and

(ii) $\big(c_t = 0 \Rightarrow \big[(c'_t, c''_t \in \{0, \hat{0}\}) \wedge (c'_t = 0 \vee c''_t = 0)\big]\big)$.

Then, evaluate the mapping A_i of X_i as follows:

`for all` $c \in \{0, \hat{0}, 1\}^{n_i}$ `do`

$$A_i(c) \leftarrow \min\{\, A_j(c') + A_k(c'') - \#_1(c) \mid c' \text{ and } c'' \text{ divide } c \,\} \tag{5.10}$$

In other words, in order to determine the value $A_i(c)$, we look up the corresponding values for coloring c in A_j (which gives us the minimum dominating set for c needed for the bags considered up to this stage in the left subtree) and in A_k (the minimum dominating set for c needed according to the right subtree), add the corresponding values, and subtract the number of "1-assignments" in c, since they would be counted twice, otherwise.

Clearly, if coloring c of node i assigns the colors 1 or $\hat{0}$ to a vertex x in X_i, we have to make sure that we use colorings c' and c'' of the children j and k that also assign the same color to x. However, if c assigns color 0 to x, it is sufficient to justify this color by *at least one* of the colorings c' or c''. Observe that, by the monotonicity of A_j and A_k we obtain the same "min" in Assignment (5.10) if we replace condition (ii) in the definition of "divide" by:

$$\text{(ii)'}\ \left(c_t = 0 \ \Rightarrow\ (c'_t, c''_t \in \{0, \hat{0}\} \wedge c'_t \neq c''_t)\right).$$

Step 3 (final evaluation). Let r denote the root of T. For the domination number $\mathrm{ds}(G)$, we finally get

$$\mathrm{ds}(G) = \min\{\, A_r(c) \mid c \in \{0, 1\}^{n_r} \,\}. \tag{5.11}$$

The minimum in Eq. (5.11) is taken only over colorings containing colors 1 and 0, since a valid dominating set does not contain "unresolved" vertices of color $\hat{0}$. Also, note that, when bookkeeping how the minima in the assignments (5.6), (5.10), and (5.11) of Step 2 and Step 3 were obtained, this algorithm constructs a dominating set D corresponding to $\mathrm{ds}(G)$.

5.3.2.3 Correctness and Time Complexity

For the correctness of the algorithm, we observe the following. Firstly, property (i) of a tree decomposition (see Definition 5.1.1) guarantees that each vertex is assigned a color. Secondly, in our initialization Step 1, as well as in the updating process for INTRODUCE NODES and JOIN NODES of Step 2, we made sure that the assignment of color 0 to a vertex x always guarantees that, at the current stage of the algorithm, x is already dominated by a vertex from previous bags. Since, by property (ii) of a tree decomposition (see Definition 5.1.1), any pair of neighbors appears in at least one bag, the validity of the colorings was checked for each such pair of neighbors at some point of the algorithm. And, thirdly, property (iii) of a tree decomposition (see Definition 5.1.1), together with the comments given in Step 2 of the algorithm, implies that the updating of each mapping is done consistently with all mappings that have been visited earlier in the algorithm.

Claim 5.3.2 *The total running time of the algorithm is* $O(4^{\ell} N)$.

Proof: It is very easy to see that the evaluations for Step 1 (Assignment (5.5)) and the evaluations for FORGET and INTRODUCE nodes in Step 2 (Assignments (5.6), (5.7), (5.8), and (5.9)) can be carried out in time $O(3^{n_i} n_i)$.

The most time-expensive step is the updating-process for a JOIN NODE. We claim that the evaluations in Assignment (5.10) can be carried out in time $O(4^{n_i})$. This basically follows from the definition of "divide." Note that the running time of this step is given by

$$\sum_{c \in \{0, \hat{0}, 1\}^{n_i}} \left|\{\, (c', c'') \mid c' \text{ and } c'' \text{ divide } c \,\}\right|. \tag{5.12}$$

For given $c \in \{0, \hat{0}, 1\}^{n_i}$, with $z := \#_0(c)$, we have 2^z many pairs (c', c'') that divide c (if we use condition (ii)' instead of condition (ii) in the definition of "divide" (sic!)). Since there are $2^{n_i - z}\binom{n_i}{z}$ many colorings c with $\#_0(c) = z$, again using condition (ii)' instead of (ii), the expression in (5.12) equates to

$$\sum_{z=0}^{n_i} 2^{n_i - z}\binom{n_i}{z} \cdot 2^z \quad = \quad 4^{n_i}.$$

The claim follows since we potentially can have $O(N)$ many JOIN NODES and since $n_i \leq \ell + 1$ for all nodes $i \in V(T)$. □

This finishes the proof for Theorem 5.3.1.

5.3.3 Further Applications and Extensions

In this subsection, we describe how our new dynamic programming strategy can be carried over to further "domination-like" problems as, e.g., treated in [181, 183, 184].

5.3.3.1 DOMINATING SET WITH PROPERTY P

We revisit the DOMINATING SET WITH PROPERTY P problem (see Subsection 1.3.4) and derive algorithms where the base σ_i of the exponential term and the number λ_i of colors needed for the mappings in the dynamic programming depend on P.

Theorem 5.3.3 *If a width ℓ and N nodes tree decomposition of a graph is known, then we can solve the subsequent problems $\mathcal{P}_i$ in time $O(\sigma_i^\ell N)$, using λ_i colors in the dynamic programming step:*

- $\mathcal{P}_1 =$ INDEPENDENT DOMINATING SET: $\sigma_1 = 4$, $\lambda_1 = 3$;
- $\mathcal{P}_2 =$ TOTAL DOMINATING SET: $\sigma_2 = 6$, $\lambda_2 = 4$;
- $\mathcal{P}_3 =$ PERFECT DOMINATING SET: $\sigma_3 = 4$, $\lambda_3 = 3$;
- $\mathcal{P}_4 =$ PERFECT CODE: $\sigma_4 = 4$, $\lambda_4 = 3$;
- $\mathcal{P}_5 =$ TOTAL PERFECT DOMINATING SET: $\sigma_5 = 6$, $\lambda_5 = 4$.

Proof: The proof is deferred to the Appendix at the end of this chapter. □

Note that our updating technique which makes strong use of the monotonicity of the mappings outperforms the results of Telle and Proskurowski. The corresponding constants σ_i' for the above listed problems that were derived in [183, Theorem 4, Table 1] and [184, Theorem 5.7] are $\sigma_1' = 9$, $\sigma_2' = 16$, $\sigma_3' = 9$, $\sigma_4' = 9$, and $\sigma_5' = 16$ (see Table 5.1 for an overview).

5.3.3.2 Weighted Versions of DOMINATING SET

Our algorithm can be adapted to the weighted version of DOMINATING SET (and its variants): Take a graph $G = (V, E)$ together with a positive integer weight function $w : V \to \mathbb{N}$. The weight of a vertex set $D \subseteq V$ is defined as $w(D) = \sum_{v \in D} w(v)$. The WEIGHTED DOMINATING SET problem is the task to determine, given a graph $G = (V, E)$ and a weight function $w : V \to \mathbb{N}$, a dominating set with minimum weight.

Only small modifications in the bookkeeping technique used in Theorem 5.3.1 (or Theorem 5.3.3) are necessary in order to solve the weighted version of DOMINATING SET (and its variations). More precisely, we have to adapt the initialization (5.5) of the mappings A_i for the bag $X_i = (x_{i_1}, \dots, x_{i_{n_i}})$ according to:

for all $c = (c_1, \dots, c_{n_i}) \in \{0, \hat{0}, 1\}^{n_i}$ do

$$A_i(c) \leftarrow \begin{cases} +\infty & \text{if } c \text{ is locally invalid for } X_i \\ w(c) & \text{otherwise,} \end{cases} \tag{5.13}$$

where $w(c) := \sum_{t=0, c_t=1}^{n_i} w(x_{i_t})$. The updating of the mappings A_i in Step 2 in the algorithm of Theorem 5.3.1 (or Theorem 5.3.3) is adapted similarly.

5.3.3.3 RED-BLUE DOMINATING SET

We finally turn our attention to the following version of DOMINATING SET, called RED-BLUE DOMINATING SET[13] which is closely related to the FACE COVER problem as defined in Subsection 1.3.4 [39, 166].

Definition 5.3.4 RED-BLUE DOMINATING SET *is the optimization problem* $(I, F, c, \min)$ *which is defined as follows:*

(i) *The set of instances* I *consists of all bipartite graphs* $G = (V_{red} \dot\cup V_{blue}, E)$.

(ii) *A feasible solution for a bipartite graph is a set* $V' \subseteq V_{red}$. *which dominates all vertices in* V_{blue}, *i.e.,* $V_{blue} \subseteq N(V')$.

(iii) *The cost for a feasible solution* V' *is given by* $c_Q(V') := |V'|$.

The following result will be used to obtain a fixed-parameter algorithm for par-FACE COVER (see Subsection 5.4.1).

Theorem 5.3.5 *Let a bipartite graph* $G = (V_{red} \dot\cup V_{blue}, E)$ *be given together with a tree decomposition of width* ℓ. *Then,* RED-BLUE DOMINATING SET *can be solved in time* $O(3^{\ell} N)$, *where* N *is the number of nodes of the tree decomposition.*

Proof: The proof is deferred to the Appendix at the end of this chapter. □

[13] Observe that RED-BLUE DOMINATING SET is *not* a variant of DOMINATING SET in the sense of the first subsection, because a solution V' is not a dominating set, since red vertices cannot and hence need not be dominated by red vertices.

5.4 Putting it all Together: $2^{O(\sqrt{k})}$-Algorithms for LSP-Problems

We now piece together the results obtained in Sections 5.2 and 5.3 to derive fixed-parameter algorithms running in time $O(2^{O(\sqrt{k})}n)$ for problems which have the Layerwise Separation Property and admit a dynamic programming approach on tree decompositions. The constant hidden in the O-notation of the exponent depends on the given problem and we will provide a formula to compute these constants from problem-specific parameters. The corresponding result is given in Subsection 5.4.1. Besides, in Subsection 5.4.2 similar results will be derived by pursuing a slightly different strategy.

5.4.1 Using Tree Decompositions

Melting our results on "Phase 1" (the construction of tree decompositions, see Section 5.2) with our results on "Phase 2" (dynamic programming on tree decompositions 5.3) yields the following main result.

Theorem 5.4.1 *Let* par-$\mathcal{G}$ *be a parameterized problem on planar graphs. Suppose that*

(i) par-$\mathcal{G}$ *has the LSP of width* w *and size-factor* d*, and*

(ii) *there exists a time* $O(\sigma^{\ell} N)$ *algorithm to solve* $\mathcal{G}$ *if the input graph* G *is given together with an* N *node tree decomposition of width* ℓ*.*

Then, there is an algorithm to solve par-$\mathcal{G}$ *in time*

$$O(\,\sigma^{3w-1} \cdot 2^{\theta_1(\sigma,d)\sqrt{k}}\,n\,), \quad \textit{where } \theta_1(\sigma, d) = 2\,(\log \sigma)\,\sqrt{6d}.$$

Proof: Given an instance (G, k), in a first phase, we use the algorithm of Corollary 5.2.25 to decide either if $(G, k) \notin$ par-$\mathcal{G}$, or compute a tree decomposition with $O(n)$ nodes and of width bounded by

$$2\sqrt{6dk} + (3w - 1).$$

This first phase takes $O(\sqrt{k}\,n)$ time.

In a second phase, we use the given tree decomposition algorithm to solve the problem in time $O(\,\sigma^{2\sqrt{6dk}+(3w-1)}\,n\,)$. □

For concrete examples, we revisit our threesome of graph problems. The algorithms in the next corollary[14] again match our lower bounds given in Theorem 4.2.10. As usual, in the following, n will always denote the number of vertices of an input graph and k will refer to the corresponding problem parameter according to Definition 1.3.2.

[14] We remark that the running times in Corollary 5.4.2 are pure worst-case upper bounds. In particular, in the case of par-DOMINATING SET there seems much room for problem-specific improvements. Also, we want to emphasize that our main goal was to pursue a general methodology with the concept of the Layerwise Separation Property. Following this aim, we deliberately sacrificed a more-fine grained, problem-specific analysis.

Corollary 5.4.2

(i) par-VERTEX COVER *on planar graphs can be solved in time* $O(2^{4\sqrt{3k}}k + kn)$.[15]

(ii) par-INDEPENDENT SET *on planar graphs can be solved in time* $O(2^{4\sqrt{6k}}k + n^2)$.[16]

(iii) par-DOMINATING SET *on planar graphs can be solved in time* $O(2^{12\sqrt{34k}}k + n^3)$.[17]

Proof: As a first step, we apply linear problem kernelization to the given input instance (see Corollary 2.1.8 for par-VERTEX COVER, Proposition 2.1.4 for par-INDEPENDENT SET, and Theorem 2.2.1 for par-DOMINATING SET). The given running times are immediate consequences of Theorem 5.4.1, if we plug in the corresponding values for size-factor and width of the LSP for par-VERTEX COVER (see Lemma 5.2.13), for par-INDEPENDENT SET (see Example 5.2.12), and for par-DOMINATING SET (see Proposition 5.2.14), and if we use the corresponding running times for the dynamic programming on tree decompositions (see Table 5.1). □

Compare the running times in Corollary 5.4.2 with the ones that were derived using the divide-and-conquer strategy based on separation (see Corollary 4.2.2 in Subsection 4.2.1). The improvements are remarkable, and, moreover, the tree decomposition based approach is applicable to a wider class of graph problems since we do not need to assume the existence of a linear problem kernel. The following remark extends the list for which fixed-parameter algorithms with a sublinear term in the exponent are possible (and for which we are *not* aware of a linear problem kernel).

Remark 5.4.3 By Corollary 5.2.17, the result of Corollary 5.4.2.(iii) extends to all variants of the par-DOMINATING SET problem that are listed in Theorem 5.3.3. More precisely, we obtain time $O(2^{O(\sqrt{k})}n)$ algorithms for the following problems: par-INDEPENDENT DOMINATING SET, par-TOTAL DOMINATING SET, par-PERFECT DOMINATING SET, par-PERFECT CODE, par-TOTAL DOMINATING SET.

We finish this subsection with a glance at the par-FACE COVER problem (see Definition 1.3.6). These considerations are meant to demonstrate that—even if we may not be able to directly prove the LSP for a graph problem—the previously established techniques can be directly used to derive a time $O(2^{O(\sqrt{k})}n)$ algorithm. We need the following auxiliary notion.

Let $G = (V, E)$ be a plane graph. Let F denote the set of faces of G. A mapping $r : F \to V$ is called a c-*bounded face representation* if and only if (i) for all $v \in r(F)$, we have $|r^{-1}(v)| \leq c$, and (ii) for all $f \in r^{-1}(v)$, v lies on the boundary of f.

Lemma 5.4.4 *Each plane graph has a* 5*-bounded face representation. Moreover, such a face representation function can be constructed in time* $O(n)$.

[15] Alternatively, not using linear problem kernelization as preprocessing, we would obtain running time $O(2^{4\sqrt{3k}}n)$.

[16] Alternatively, using the five-coloring algorithm (see Remark 2.1.5) as a base of the linear problem kernel instead of a four-coloring, the running time for the preprocessing is linear instead of quadratic, but the size-factor of the LSP on the set of reduced instances is five instead of four. Hence, we would obtain running time $O(2^{2\sqrt{30k}}k + n)$.

[17] Alternatively, not using linear problem kernelization as preprocessing, we obtain running time $O(2^{12\sqrt{34k}}n)$.

Proof: The proof is deferred to the Appendix at the end of this chapter. □

Proposition 5.4.5 par-FACE COVER *can be solved in time* $O(3^{36\sqrt{34k}}n)$.

Proof: Let $(G = (V, E), \phi)$ be a plane graph with face set F. Due to Lemma 5.4.4, we can find a 5-bounded face representation $r : F \to V$ in time $O(n)$.

Consider the following graph: Add a new vertex in the interior of each face of G, and make each such "face vertex" adjacent to all vertices that are on the boundary of that face. These are the only edges of the bipartite graph $G' = (V', E')$. Write $V' = V_F \cup V$, where V_F is the set of face vertices, i.e., each $v \in V_F$ represents a face f_v in G. In other words, V and V_F form the bipartition of G'. Observe that G' can be viewed as an instance of RED-BLUE DOMINATING SET: the face vertices V_F are "red" and the other vertices V are "blue." Now, we would like to apply Theorem 5.3.5 on a suitable tree decomposition of G' to finish the proof.

To this end, consider the graph $\hat{G} = (V, \hat{E})$ obtained from G' by contracting each edge connecting a face vertex $v \in V_F$ of G' with $r(f_v)$. In other words, $\hat{G}$ is obtained from G by firstly removing all "original" edges from E, and then inserting, for every $f \in F$, edges between $r(f)$ and each other vertex v adjacent to f in G.

Assume that G has a face cover $C \subseteq F$ of size k. Then, $r(C)$ is a dominating set of $\hat{G}$ of size $k' \leq k$. Hence, by Corollary 5.2.26.(ii), $\mathrm{tw}(\hat{G}) \leq 6\sqrt{34k} + 8$, and a corresponding tree decomposition can be found in time $O(\sqrt{k}\,n)$. In order to be able to apply Theorem 5.3.5, we observe that a tree decomposition of G' of width at most $36\sqrt{34k} + 48$ can be obtained from the tree decomposition of $\hat{G}$ by enhancing the bags of $\hat{G}$'s tree decomposition according to the following rule: if $r(f)$ is in some bag for some face $f \in F$, then put all $v \in V_F$ into that bag which satisfies $r(f_v) = r(f)$. The reader should verify that this, indeed, yields a tree decomposition of G', and the claimed width bound follows from Lemma 5.4.4. □

5.4.2 Using Bounded Outerplanarity

We sketch an alternative approach to obtain time $O(2^{O(\sqrt{k})}n)$ algorithms for LSP-problems in a slightly different context. The presentation here will be informal and the reader is referred to [4] for details.

Recall that in the approach followed in the previous subsection, the LSP was used to derive an appropriate partial layerwise separation of a given input graph. From this partial layerwise separation, a tree decomposition was computed (see Theorem 5.2.23), on which we performed a dynamic programming strategy. We now use an appropriate partial layerwise separation for a direct dynamic programming without doing the detour via tree decompositions.

Assume that the underlying graph problem $\mathcal{G}$ is a vertex selection problem which is glueable with λ colors (see Definition 4.1.10 in Subsection 4.1.2.1). Very informally speaking, this means that if a graph instance $G = (V, E)$ is given together with a separation (V_A, V_S, V_B), a divide-and-conquer algorithm can check all possible assignments of the λ colors to the vertices in V_S and then combine the optimal solutions for $G[V_A \cup V_S]$ and $G[V_B \cup V_S]$ in a suitable manner. Suppose we are given a partial layerwise separation $\mathcal{S} = (S_1, \ldots, S_q)$ (see Definition 5.2.8) with corresponding graph chunks $\mathcal{C}_{\mathcal{S}} = (G_0, \ldots, G_q)$. The glueability assumption then makes it possible to pursue a dynamic programming which—by sweeping over the separators $S_1, \ldots, S_q$—successively compute (partial) solutions of $\mathcal{G}$ on the graphs $G^i := G[V(G_0) \cup S_1 \cup \cdots \cup S_{i-1} \cup V(G_{i-1}) \cup S_i]$ (for $1 \leq i \leq q$). Carrying out this method in detail, we obtain the following result.

Theorem 5.4.6 *Let $\mathcal{G}$ be a vertex selection problem on planar graphs. Suppose that*

(i) par-$\mathcal{G}$ *has the LSP of width* w *and size-factor* d,

(ii) $\mathcal{G}$ *is glueable with* λ *colors, and*

(iii) *there exists an algorithm that solves* $\mathcal{G}$ *for a given precolored graph* G *in time* $O(\tau^{\mathrm{out}(G)} n)$.

Then, there is an algorithm to solve par-$\mathcal{G}$ *in time*

$$O(\tau^{w} \cdot 2^{\theta_2(\lambda,\tau,d)\sqrt{k}}\, n), \quad \text{where } \theta_2(\lambda, \tau, d) = 2\sqrt{2d \log(\lambda) \log(\tau)}.$$

Proof: A sketch of the proof is given in the Appendix at the end of the Chapter. □

A source for algorithms that fulfill condition (iii) in Theorem 5.4.6 is given in [31].

Which of the two algorithms (presented in this and the previous subsection, respectively) on layerwisely separated graphs should be preferred? For that purpose, let us compare the results obtained in Theorems 5.4.1 and 5.4.6. Clearly, both results rely on the LSP assumption. Besides that, Theorem 5.4.1 requires the existence of a linear time algorithm for bounded treewidth graphs, whereas in Theorem 5.4.6 one needs (besides the glueability assumption) the existence of a linear time algorithm for graphs with bounded outerplanarity. The interrelation between these assumptions is given by the following observation, which is a consequence of Theorem 5.2.4.

Lemma 5.4.7 *Let $\mathcal{G}$ be a problem on planar graphs. Suppose that there exists a time $O(\sigma^{\ell} n)$ algorithm that solves $\mathcal{G}$ for a (precolored) graph that is given together with a tree decomposition of width ℓ. Then, there is an algorithm that solves $\mathcal{G}$ for a (precolored) graph G in time $O(\tau^{\mathrm{out}(G)} n)$ for $\tau = \sigma^3$.* □

Using this result we obtain a version of Theorem 5.4.6 that is directly comparable to Theorem 5.4.1.

Theorem 5.4.8 *Let $\mathcal{G}$ be a vertex selection problem on planar graphs. Suppose that*

(i) par-$\mathcal{G}$ *has the LSP of width* w *and size-factor* d,

(ii) $\mathcal{G}$ *is glueable with* λ *colors, and*

(iii) *there exists a time $O(\sigma^{\ell} N)$ algorithm to solve $\mathcal{G}$ for a precolored graph G if G is given together with an N node tree decomposition of width ℓ.*

Then, there is an algorithm to solve par-$\mathcal{G}$ *in time*

$$O(\sigma^{3w} \cdot 2^{\theta_3(\lambda,\sigma,d)\sqrt{k}} n), \quad \text{where } \theta_3(\lambda, \sigma, d) = 2\sqrt{6d \log(\lambda) \log(\sigma)}.$$

The exponential factor of the algorithm in Theorem 5.4.8, i.e., $\theta_3(\lambda, \sigma, d)$, is related to the corresponding exponent of Theorem 5.4.1, i.e., $\theta_1(\sigma, d)$ in the following way:

$$\sqrt{\log \lambda} \cdot \theta_1(\sigma, d) = \sqrt{\log \sigma} \cdot \theta_3(\lambda, \sigma, d).$$

From this we derive that

- if $\lambda > \sigma$, the algorithm in Theorem 5.4.1 outperforms the result of Theorem 5.4.8,
- if $\lambda < \sigma$, the algorithm in Theorem 5.4.8 outperforms the result of Theorem 5.4.1.

However, in order to apply Theorem 5.4.8, we need the two extra assumptions that we have a glueable vertex selection problem and that we can deal with precolored graphs in the treewidth algorithm.

Remark 5.4.9 Whereas, for par-VERTEX COVER and par-INDEPENDENT SET on planar graphs, we have $\lambda = \sigma = 2$ and, hence, the approaches in Theorem 5.4.1 and Theorem 5.4.8 yield the same worst-case upper bounds, the situation is slightly different for par-DOMINATING SET on planar graphs: Here, we have $\sigma = 4$. On the other hand, we already argued that—for algorithmic purposes—the constant $\lambda = 4$ (see Lemma 4.1.11) can be dropped to $\lambda = 3$ (see Remark 4.1.12). All in all, Theorem 5.4.8 gives us an algorithm running in time

$$O(\, 2^{12\sqrt{17 \cdot \log(3) \cdot k}}\, n\,)$$

for par-DOMINATING SET on planar graphs, which slightly beats the $O(2^{12\sqrt{34k}}n)$ from Corollary 5.4.2.(iii). Alternatively, if we use the linear problem kernelization for par-DOMINATING SET on planar graphs (see Theorem 2.2.1) as a preprocessing, the running time of the algorithm becomes $O(2^{12\sqrt{17 \cdot \log(3) \cdot k}}k + n^3)$.

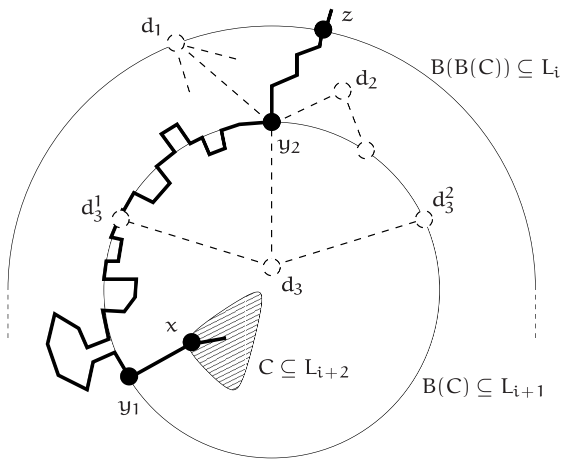

Figure 5.7: S_i separates layer L_{i-1} from layer L_{i+3}.

Appendix

Proof of Lemma 5.1.8:
W.l.o.g. we assume G is connected. We make the assumption that with the embedding, for each vertex v, the incident edges of v are given in clockwise order as they appear in the embedding. Most (linear time) graph planarity testing and embedding algorithms yield such orderings of the edge lists (see [72]). With the help of these orderings, one can build, in $O(|V|)$ time, the dual graph G^*, with pointers from edges of G to the two adjacent faces in the dual. We first partition the set of faces into 'layers': put a face f in layer L^*_{i+1} if the distance of this face in the dual graph to the exterior face is i. This distance can be determined in linear time using breadth-first search on the dual graph.

Now, a vertex v of G belongs to layer L_i for the smallest i such that v is adjacent to a face f_v in L^*_i. Note that faces can belong to layer L^*_{r+1}, but not to layers L^*_s with $s > r + 1$. □

Proof of Lemma 5.2.16:
We prove the three properties of Definition 5.2.8 separately. Recall the construction of S_i via lower, middle, and upper triples.

ad property (i): By construction, it is clear, that $S_i \subseteq L_i \cup L_{i+1} \cup L_{i+2}$.

ad property (ii): We have to show that S_i separates vertices of layers L_{i-1} and L_{i+3}. Suppose there is a path P (with no repeated vertices) from layer L_{i+3} to layer L_{i-1} that avoids S_i. This clearly implies that there exists a path P' from a vertex x in a layer component C of layer L_{i+2} which is not a leaf in the layer decomposition tree to a vertex $z \in B(B(C))$ in layer L_i which has the following two properties (see Fig. 5.7):

- $P' \cap S_i = \emptyset$.
- All vertices between x and z belong to layer L_{i+1}.

This can be achieved by simply taking a suitable subpath P' of P. Let y_1 (and y_2, respectively) be the first (last) vertex along the path P' from x to z that lies on the boundary cycle $B(C) \subseteq L_{i+1}$.

Obviously, y_2 cannot be an element of D since, then, it would appear in a middle triple of layer L_i and, hence, in S_i. We now consider the vertex that dominates y_2. This vertex can lie in layer L_i, L_{+1}, or L_{i+2}.

Suppose first that y_2 is dominated by a vertex $d_1 \in D \cap L_i$. Then, d_1 is in $B(B(C))$, simply by definition of the boundary cycle (see Fig. 5.7). Since G is planar, this implies that y_2 must be an "outermost" neighbor of d_1 among all elements in $N(d_1) \cap B(C)$. If this were not the case, then there would be an edge from d_1 to a vertex on $B(C)$ that leaves the closed region bounded by $\{d_1, y_2\}$, the path from y_2 to z, and the corresponding path from z to d_1 along $B(B(C))$. Hence, y_2 is in the upper triple of layer L_i which is associated with d_1. This contradicts the assumption that P' avoids S_i.

Now, suppose that y_2 is dominated by a vertex $d_2 \in D \cap L_{i+1}$ (see Fig. 5.7). By definition of the middle triples, this clearly implies that y_2 is in the middle triple associated with d_2. Again, this contradicts the fact that $P' \cap S_i = \emptyset$.

Consequently, the dominating vertex d_3 of y_2 has to lie in layer L_{i+2}. Let $\{d_3, d_3^1, d_3^2\}$, where $d_3^1, d_3^2 \in N(d_3) \cap B(C)$, be the lower triple associated with d_3 and the component C (see Fig. 5.7). By definition of a lower triple, C is contained in the region enclosed by $\{d_3^1, d_3\}, \{d_3, d_3^2\}$, and the path from d_3^2 to d_3^1 along $B(C)$, which—assuming that $y_2 \notin \{d_3, d_3^1, d_3^2\}$—does not hit y_2 (see Fig. 5.7). We now observe that, whenever the path from y_1 to y_2 leaves the cycle $B(C)$ to its exterior, say at a vertex q, then it has to return to $B(C)$ at a vertex $q' \in N(q) \cap B(C)$. Otherwise, we would violate the fact that, by definition, $B(C) \subseteq L_{i+1}$. This, however, shows that the path P' has to hit either d_3^1 or d_3^2 on its way from y_1 to y_2. Since $d_3^1, d_3^2 \in S_i$, this case also contradicts the fact that $P' \cap S_i = \emptyset$.

ad property (iii): We will show that $\sum_{i=1}^{r} |S_i| \leq 51k$. For this purpose, let $k_i := |D \cap L_i|$, and let c_i denote the number of layer components in layer L_i which are not a leaf in the layer decomposition tree (i.e., which have vertices from layer L_{i+1} inside). Our first claim is that

$$|S_i| \leq 5(k_i + k_{i+1} + k_{i+2}) + 12c_{i+2}. \tag{5.14}$$

To see this, we give bounds for the number of vertices in upper, middle, and lower triples, separately.

Firstly, we discuss the upper triples of layer i, which are associated with a vertex $x \in D \cap L_i$ and a layer component C of layer L_{i+2} (that has further vertices from L_{i+3} inside). Consider the bipartite graph G' which has vertices for each such layer component C in layer L_{i+2} and for each vertex in $D \cap L_i$. Whenever a vertex in $D \cap L_i$ has a neighbor in $B(C)$, an edge is drawn between the corresponding vertices in G'. Each edge in G', by construction, may correspond to an upper triple of layer L_i. Note that G' is a planar bipartite graph whose bipartition subsets consist of k_i and c_{i+2} vertices, respectively. Thus, the number of edges of G' is linear in the number of vertices; more precisely, it is bounded by $2(k_i+c_{i+2})$ (see [159, Corollary 1.2.]). From this, we obtain an upper bound for the number of vertices in upper triples of layer L_i as follows: Potentially, each vertex of $D \cap L_i$ appears in an upper triple and, for each edge in G', we possibly obtain two further vertices in an upper triple. This shows that the total number of vertices in upper triples is bounded by $k_i + 4(k_i + c_{i+2})$.

A similar analysis can be used to show that the number of vertices in the lower triples is bounded by $k_{i+2} + 4(k_{i+2} + c_{i+2})$ and that the number of vertices in the middle triples can be bounded by $k_{i+1} + 4(k_{i+1} + c_{i+2})$. By definition of S_i, this proves the claim in Eq. 5.14.

Our second claim is that, for the number c_i of layer components in layer i which are not leaves in the layer decomposition tree, it holds

$$c_i \leq k_i + k_{i+1} + k_{i+2}. \tag{5.15}$$

To see this, observe that there is at least one vertex of layer L_{i+1} contained within each such layer component. Such a vertex can only be dominated by a vertex from layer L_i, L_{i+1}, or L_{i+2}. In this way, we get the claimed upper bound.

Combining Equations (5.14) and (5.15) and using the fact that $\sum_{i=1}^{r} k_i = k$, some simple arithmetic shows that $\sum_{i=1}^{r} |S_i| \leq 51k$. □

Proof (Sketch) of Theorem 5.3.3:
For problem $\mathcal{P}_1$, the algorithm is identical to the one in Theorem 5.3.1 (see Subsection 5.3.2.2) except for a slight modification. In contrast to the algorithm given in the proof of Theorem 5.3.1, after each update of a mapping A_i for bag X_i, we check, for each coloring $c \in \{0, \hat{0}, 1\}^{n_i}$, if there exist two vertices $x, y \in X_i$ that both are assigned color 1 by c, and, if so, set $A_i(c) \leftarrow +\infty$.

For problem $\mathcal{P}_2$, one must also distinguish for the vertices in the domination set whether or not they have been dominated by other vertices from the dominating set. We may use 4 colors:

- 1, meaning that the vertex is in the dominating set and it is already dominated;
- $\hat{1}$, meaning that the vertex is in the dominating set and it still needs to be dominated;
- 0, meaning that the vertex is not in the dominating set and it is already dominated;
- $\hat{0}$, meaning that the vertex is not in the dominating set and it still needs to be dominated.

The partial ordering $\prec$ on $C := \{0, \hat{0}, 1, \hat{1}\}$, according to which our mappings will be monotonous, is given by $\hat{1} \prec 1$, $\hat{0} \prec 0$, and $d \prec d$ for all $d \in C$.

The various steps of the algorithm for updating the mappings are similar the ones given in the algorithm of Theorem 5.3.1 (see Subsection 5.3.2.2). The most cost-expensive part again is a JOIN NODE. Here, in the Assignment (5.10), we have to adapt the definition of "divide" as follows: For a coloring $c = (c_1, \ldots, c_{n_i}) \in \{0, \hat{0}, 1, \hat{1}\}^{n_i}$ for X_i, we say that the two colorings $c' = (c'_1, \ldots, c'_{n_i})$, $c'' = (c''_1, \ldots, c''_{n_i}) \in \{0, \hat{0}, 1, \hat{1}\}^{n_i}$ *divide* c if

(i) $\big(c_t = 0 \;\Rightarrow\; (c'_t, c''_t \in \{0, \hat{0}\} \wedge c'_t \neq c''_t)\big)$, and

(ii) $\big(c_t = 1 \;\Rightarrow\; (c'_t, c''_t \in \{1, \hat{1}\} \wedge c'_t \neq c''_t)\big)$.

Similar to the proof of Claim 5.3.2 the running time for updating a JOIN NODE is given by

$$\sum_{c\in\{0,\hat{0},1,\hat{1}\}^{n_i}} \big|\{\,(c',c'')\mid c' \text{ and } c'' \text{ divide } c\,\}\big|. \tag{5.16}$$

We use a combinatorial argument to compute this expression. For a fixed coloring $c \in \{0,\hat{0},1,\hat{1}\}^{n_i}$, we have $\#_0(c) \in \{0,\ldots,n_i\}$, and $\#_1(c) \in \{0,\ldots,n_i - \#_0(c)\}$. The number of colorings $c \in \{0,\hat{0},1,\hat{1}\}^{n_i}$ with $\#_0(c) = z_0$ and $\#_1(c) = z_1$ is given by $2^{(n_i-z_0-z_1)}\binom{n_i}{z_0}\binom{n_i-z_0}{z_1}$. Since, by definition of "divide," for each position in c with $c_t = 0$ or $c_t = 1$, we have to consider two different divide pairs, we get

$$\begin{aligned}
&\sum_{c\in\{0,\hat{0},1,\hat{1}\}^{n_i}} \big|\{\,(c',c'')\mid c' \text{ and } c'' \text{ divide } c\,\}\big| \\
&= \sum_{\#_0(c)=0}^{n_i}\ \sum_{\#_1(c)=0}^{n_i-\#_0(c)} 2^{\#_0(c)}\,2^{\#_1(c)}\left(2^{(n_i-\#_0(c)-\#_1(c))}\binom{n_i}{\#_0(c)}\binom{n_i-\#_0(c)}{\#_1(c)}\right) \\
&= 2^{n_i}\sum_{\#_0(c)=0}^{n_i}\binom{n_i}{\#_0(c)}\,2^{n_i-\#_0(c)} \qquad = \quad 6^{n_i}
\end{aligned}$$

This determines the running time of the algorithm.

For problem $\mathcal{P}_3$, we again use the algorithm given in the proof of Theorem 5.3.1 (see Subsection 5.3.2.2) except for the following modification in the semantics of the color sets. A vertex is colored "grey" if it is dominated by exactly one "black" vertex which either lies in the "current" bag of the tree decomposition algorithm or in one of its child bags. It is easy to see how to adapt the updating process in a suitable manner.

For problem $\mathcal{P}_4$, and problem $\mathcal{P}_5$, respectively, we can use appropriate combinations of the arguments for problems ($\mathcal{P}_1$, $\mathcal{P}_3$), and problems ($\mathcal{P}_2$, $\mathcal{P}_3$), respectively. □

Proof (Sketch) of Theorem 5.3.5:
Basically, the technique exhibited in Theorem 5.3.1 (see Subsection 5.3.2.2) can be applied. Due to the bipartite nature of the graph, only two "states" have to be stored for each vertex: red vertices are either within the dominating set or not (represented by colors 1_{red} and 0_{red}, respectively), and blue vertices are either already dominated or not yet dominated (represented by colors 0_{blue} and $\hat{0}_{\text{blue}}$, respectively).

We consider our bags as bipartite sets, i.e.,

$$X_i := X_{i,\text{red}} \cup X_{i,\text{blue}},$$

where $X_{i,\text{red}} := X_i \cap V_{\text{red}}$ and $X_{i,\text{blue}} := X_i \cap V_{\text{blue}}$. Let $n_{i,\text{red}} := |X_{i,\text{red}}|$ and $n_{i,\text{blue}} := |X_{i,\text{blue}}|$, i.e., $|X_i| =: n_i = n_{i,\text{red}} + n_{i,\text{blue}}$.

The partial ordering $\prec$ on the color set $C = C_{\text{red}} \cup C_{\text{blue}}$, where $C_{\text{red}} := \{1_{\text{red}}, 0_{\text{red}}\}$ and $C_{\text{blue}} := \{0_{\text{blue}}, \hat{0}_{\text{blue}}\}$, is given by $\hat{0}_{\text{blue}} \prec 0_{\text{blue}}$ and $d \prec d$ for all $d \in C$.

A *valid* coloring for X_i is a coloring where we assign colors from C_{red} to vertices in $X_{i,\text{red}}$ and colors from C_{blue} to vertices in $X_{i,\text{blue}}$. The various steps of the algorithm for updating the mappings are similar to the ones given in the algorithm of Theorem 5.3.1 (see Subsection 5.3.2.2).

Again, the most cost-expensive part is the updating of a JOIN NODE. For a correct updating of JOIN NODES, we adapt the definition of "divide" that appears in the Assignment (5.10) according to: For a valid coloring $c = (c_1, \ldots, c_{n_i}) \in C^{n_i}$ of X_i, we say that the valid colorings $c' = (c'_1, \ldots, c'_{n_i})$, $c'' = (c''_1, \ldots, c''_{n_i}) \in C^{n_i}$ *divide* c if

(i) $\big(c_t \neq 0_{\text{blue}} \;\Rightarrow\; (c'_t, c''_t = c_t)\big)$, and

(ii) $\big(c_t = 0_{\text{blue}} \;\Rightarrow\; (c'_t, c''_t \in \{0_{\text{blue}}, \hat{0}_{\text{ blue}}\} \wedge c'_t \neq c''_t)\big)$.

For a fixed valid coloring c that contains $z := \#_{0_{\text{blue}}}(c)$ many colors 0_{blue}, the number of pairs that divide c is 2^z. Since there are $2^{n_{i,\text{red}}}\binom{n_{i,\text{blue}}}{z}$ many colorings with $\#_{0_{\text{blue}}}(c) = z$, the total number of pairs that divide a fixed coloring c is upperbounded by

$$\sum_{z=0}^{n_{i,\text{blue}}} 2^{n_{i,\text{red}}}\binom{n_{i,\text{blue}}}{z} \cdot 2^z \quad = \quad 2^{n_{i,\text{red}}} \cdot 3^{n_{i,\text{blue}}} \quad \leq \quad 3^{n_i}.$$

This determines the running time of the algorithm. Note that in the worst-case, for a bag X_i, we may have $n_{i,\text{red}} = 0$ and $n_{i,\text{blue}} = n_i$. □

Proof of Lemma 5.4.4:
Let $(G_0 = (V_0, E_0), \phi_0)$ be a plane graph. We are going to define a 5-bounded face representation r of G_0 in a step-by-step fashion.

Since G_0 is planar, there exists a vertex v_0 of degree at most five, hence, there are at most five faces adjacent to v_0. For all these faces f, we define $r(f) = v_0$. Consider, then, the graph $G_1 = (V_1, E_1)$, where $G_1 = G_0 - v_0$. We assume the "same" planar embedding for G_1 as for G_0. As before, we can find a vertex $v_1 \in V_1$ with at most five adjacent faces (in G_1). Therefore, v_1 has at most five adjacent faces in G_0 to which no vertices have yet been assigned. For all these faces f, we define $r(f) = v_1$.

Inductively, G_{i+1} is obtained as $G_i - v_i$, where v_i has degree of at most five in G_i. Here, v_i represents all adjacent faces in G_0 which are not already represented by the previously selected vertices $v_0, \ldots, v_{i-1}$. This loop is repeated until all faces f of G_0 have one representing vertex $r(f)$, i.e., until r is completely defined.

We obtain time bound $O(n)$ as follows. Maintain for every vertex its degree and maintain a data structure that contains all vertices of degree at most 5. Now, repeatedly, pick a vertex v from the data structure, and take that as the vertex in the iteration; update the degrees of its neighbors ($O(1)$ work) and add these to the data structure if their degree is at most 5. □

Proof (Sketch) of Theorem 5.4.6:
We only give a sketch of the algorithm, for a fully detailed proof we refer to [4, Section 5.2]. The overall structure of the algorithm is as follows:

- Compute some planar embedding ϕ of G.
- Find a partial layerwise separation $\mathcal{S}(\psi) = (S_1, \ldots, S_q)$ of (G, ϕ) for some suitable trade-off parameter ψ (to be determined later) (see Proposition 5.2.21). Let $\mathcal{C}_{\mathcal{S}(\psi)} = \{G_0, \ldots, G_q\}$

be the graph chunks obtained by this separation, and let $G^i := G[V(G_0) \cup S_1 \cup \cdots \cup S_{i-1} \cup V(G_{i-1}) \cup S_i]$ $(0 \leq i \leq q)$.

- For each separator S_i, we now keep a table A_i of size $\lambda^{|S_i|}$. In this table we store, for all possible assignments of the λ colors to vertices of S_i, an optimal solution for G^i with the vertices in S_i being precolored according to their current color assignment.
  - Use the given algorithm $\mathcal{A}$ to compute the table A_1 for S_1.
  - For $i := 2, \ldots, q$ do:
    - For all possible assignments of the λ colors to vertices of S_i compute the table entry corresponding to this color assignment for the table A_i as follows:
      - For all possible assignments of the λ colors to vertices in S_{i-1} do:
        - use algorithm $\mathcal{A}$ to compute an optimal solution for $G[S_{i-1} \cup V(G_{i-1}) \cup S_i]$ (where the vertices in S_{i-1} and S_i are precolored according to their current color assignment).
        - combine this optimal solution with the optimal solution for G^{i-1} under the current color-assignment for S_{i-1} (which is stored in table A_{i-1}).
  - The optimal solution is obtained as the optimum under the following computations:
    - For all possible assignments of the λ colors to vertices in S_q do:
      - use algorithm $\mathcal{A}$ to compute an optimal solution for $G[S_q \cup V(G_q)]$ (the vertices in S_q being precolored according to their current color assignment).
      - combine the optima obtained in this manner with the optimal solution for G^q under the current color assignment.

The correctness of this algorithm can be proven by an inductive argument using the formulation of glueability. The running time, for updating the entries in table A_i is bounded by

$$\lambda^{|S_i|} \cdot \lambda^{|S_{i-1}|} \cdot \tau^{\mathrm{out}(G[S_{i-1} \cup V_{G_{i-1}} \cup S_i])} \cdot n,$$

simply because, for all possible assignments of the λ colors to the vertices in S_{i-1} and S_i, we run algorithm $\mathcal{A}$ on the (precolored) graph $G[S_{i-1} \cup V_{G_{i-1}} \cup S_i]$. Hence, the total running time of the algorithm is bounded by $O(2^{\theta(\psi)} n)$, where

$$\begin{aligned}
\theta(\psi) &\leq 2\log(\lambda) \max_{i=1,\ldots,q} |S_i| \quad + \quad \log(\tau) \max_{i=1,\ldots,q} \mathrm{out}(G[S_{i-1} \cup V(G_{i-1}) \cup S_i]) \\
&\leq 2\log(\lambda)\, \psi \sqrt{dk} \quad + \quad \log(\tau) \left(\frac{\sqrt{dk}}{\psi} + w \right) \\
&= \left(2\log(\lambda)\psi + \frac{\log \tau}{\psi} \right) \sqrt{dk} \quad + \quad \log(\tau) w
\end{aligned}$$

This upper bound is minimized for $\psi = \sqrt{\log(\tau)/(2\log(\lambda))}$, which gives us the claimed value $\theta_2(\lambda, \tau, d) = 2\sqrt{2d\log(\lambda)\log(\tau)}$ and the constant τ^w for the running time. $\square$

Chapter 6

Experimental Studies

In this chapter, we demonstrate that the algorithms designed and analyzed in the previous chapters provide a valuable way of exactly solving various NP-hard problems on planar graphs in practice. The focus of the experimental study lies on the $2^{O(\sqrt{k})}$-algorithms based on tree decompositions[1] (see Chapter 5) and on the application potential of combining these with data reduction by preprocessing (see Chapter 2). Our investigations are based on a software package that implements these methods. Besides we experimented with the search tree method based on the degree-branching strategy (see Chapter 3). In a first Section 6.1, we present the software package (for the tree decomposition based approach). Section 6.2 discusses a first serious round of empirical studies with this package, demonstrating the practical usefulness of the tree decomposition concept for, e.g., par-VERTEX COVER on planar graphs. In Section 6.3, we report on the evaluation of the degree-branching search tree for par-DOMINATING SET. Finally, the impressive and probably underestimated power of problem kernelization as an extremely useful data reduction tool to speed-up algorithms is studied in Section 6.4.

Our findings, in general, show that the average-case behavior is much better than what is indicated by the worst-case bounds given in the theoretical analysis. We used combinatorial random graphs as test instances. It is important to note that all observations made in this setting do not necessarily carry over to other applications using real-world data arising from various sources. However, the results of our tests indicate that there are grounds of evidence that the algorithms have a big potential for practical applications.

6.1 The FPT-Toolbox for Planar Graph Problems

Based on LEDA [145], we designed and implemented a software package for exactly solving NP-hard problems on planar graphs. More precisely, this package offers algorithms for parameterized

[1] The reasons for favoring the $2^{O(\sqrt{k})}$-algorithms derived from tree decompositions to the algorithms based on graph separation (see Chapter 4) are twofold. Firstly, the tree decomposition technique applies to a wider range of problems (namely, all problems that have the Layerwise Separation Property). And, secondly, our worst-case analysis already suggested that the potential of tree decomposition based algorithms is higher than the one of separation techniques.

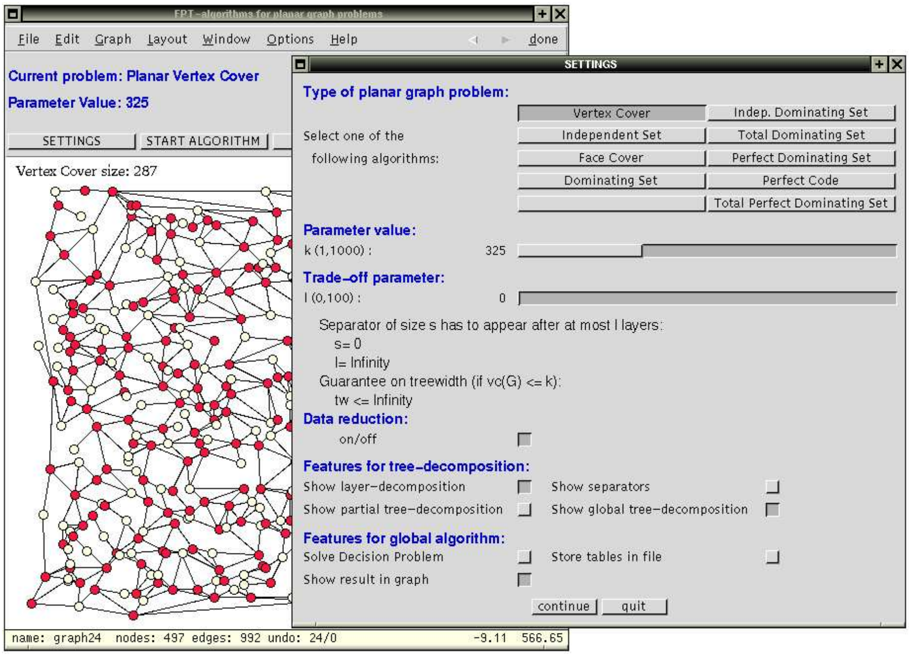

Figure 6.1: Screenshot of the graphical user interface that allows to adjust the settings of the "FPT-Toolbox."

graph problems that fit into the framework of Chapter 5, i.e., that have the so-called "Layerwise Separation Property." These include par-VERTEX COVER, par-INDEPENDENT SET, par-DOMINATING SET, par-FACE COVER, and variations thereof, such as par-INDEPENDENT DOMINATING SET, par-PERFECT DOMINATING SET, par-TOTAL DOMINATING SET, or par-PERFECT CODE.

6.1.1 Design and Use

The usage of the package is fairly easy. The program interacts with the user through a graphical user interfaces. We provide a panel for the various settings of the algorithm (see Fig. 6.1 for a screenshot). The user selects the type of problem that (s)he wants to solve among the list of graph problems which have the Layerwise Separation Property (LSP). Then, the parameter value k is chosen (i.e., the size of the desired vertex cover, independent set, dominating set, etc. the algorithm is checking for). In addition, we leave it to the user to adjust the trade-off parameter ψ (described in Remark 5.2.22). Besides, some extra optional features for the output can be adjusted. For example, it is possible to trace the algorithm by asking for outputs of the layer decomposition, the "layerwise" separators, the tree decomposition, or the tables of the dynamic programming.

The planar input graph can be drawn either directly or may be given as a file. The algorithm then solves the problem, i.e., it outputs that there is no optimal solution of size at most k (for minimization problems) or at least k (for maximization problems), or it computes an optimal solution and highlights the set of vertices corresponding to it. An example for par-VERTEX

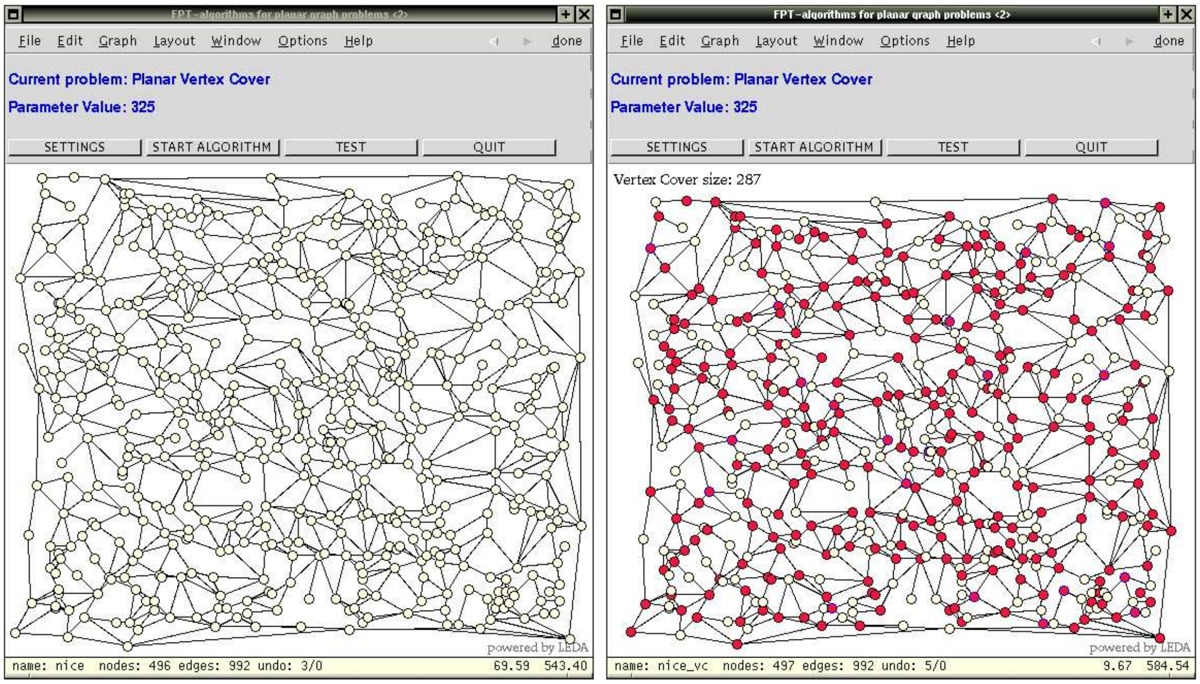

Figure 6.2: The "FPT-Toolbox," a software package for hard planar graph problems. The diagram illustrates an example for par-VERTEX COVER. The user provides the input graph (left-hand diagram) and the algorithm outputs an optimal vertex cover (right-hand diagram).

COVER is given in Fig. 6.2.

Assume that the parameterized graph problem the user wishes to solve has the LSP of width w and size-factor s. Recall from Chapter 5 that the tree decomposition based algorithm that solves the problem proceeds in two phases according to the following scheme:

Phase 1: Compute a tree decomposition for the given graph (according to Corollary 5.2.25):

(i) compute the layer decomposition of the graph (see Lemma 5.1.8).

(ii) find a layerwise separation $\mathcal{S}(\psi)$ (see Proposition 5.2.21), where

 (a) each separator $S \in \mathcal{S}(\psi)$ has size at most $S := \psi\sqrt{dk}$, and

 (b) each graph chunk has at most $(\sqrt{dk}/\psi) + w$ many layers.

 if such a layerwise separation does not exist, we can answer "no."

(iii) construct a tree decomposition for each graph chunk (see Lemma 5.1.8).

(iv) merge the tree decomposition for the graph chunks to a global tree decomposition (see Proposition 5.2.3).

Phase 2: Use problem-specific dynamic programming to solve the problem on the given tree decomposition (according to the scheme given in Subsection 5.3.1.1).

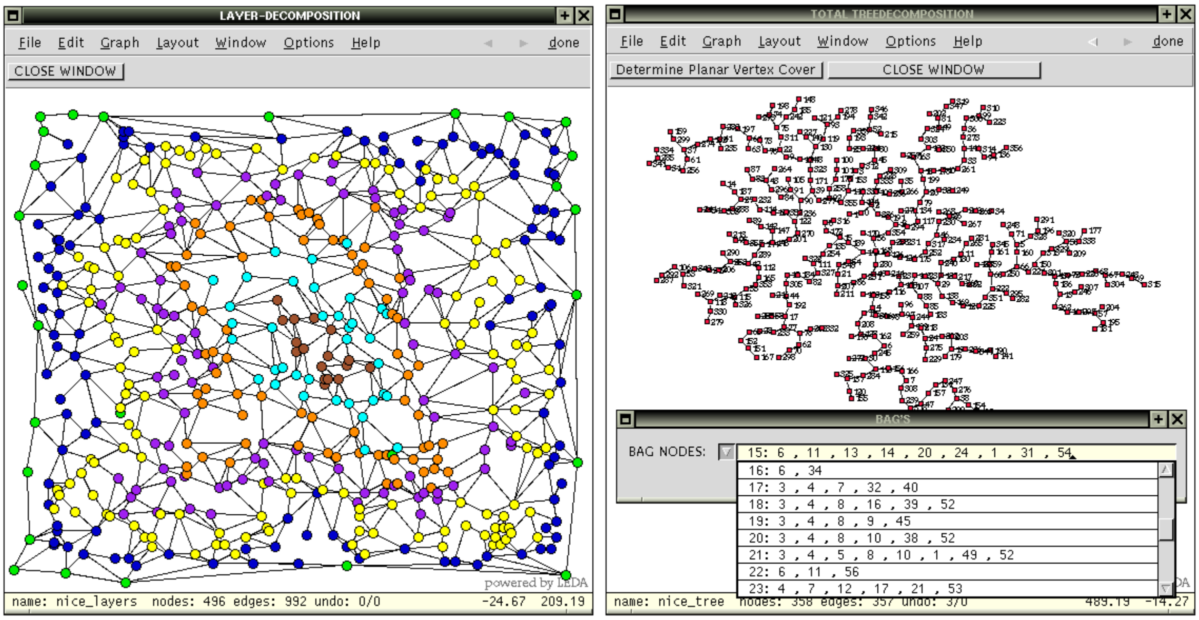

Figure 6.3: The left-hand diagram shows the layer decomposition of the input graph from Fig. 6.2. The right-hand diagram shows a tree decomposition for this graph. The tree is displayed in the upper part of the window. The bags are displayed in the extra window. There, each line corresponds to a single bag. The number in front of the semicolon of each line refers to a tree node corresponding to this bag. The numbers following the semicolon represent the graph vertices that belong to the bag.

Phase 1 of the algorithm, i.e., the construction of a tree decomposition of the given plane graph, is common to all problems offered by our software package. Phase 2 of the algorithm, i.e., the dynamic programming, is a problem-specific step.

As already mentioned above, the user can adjust the setting in such a way that the partial results, e.g., the layer decomposition of the tree decomposition, are displayed by the program. Examples for a layer decomposition and for a tree decomposition that were computed by the software package are depicted in Fig. 6.3.

6.1.2 Implementation

Our software package consists of more than 5000 lines of C++ code based on the LEDA package [145] ((non-commercial) version 4.2). Most of the implementation work was done by Frederic Dorn in a student project under my supervision [78].

The implementation is done in an object-oriented way and offers, among others, the following main procedures (for a manual of the package, we refer to [78]). Concerning the tree decomposition phase, we have, e.g.,

- `layer_decomposition(GraphWin& gw)`, creating a layer decomposition of a plane graph given in an object `GraphWin& gw` of LEDA (see Lemma 5.1.8);

- `separator(GraphWin& gw)`, determining "layerwise separation" of a plane graph, which depends on the parameter value `k`, the trade-off parameter and the specified graph problem (see Proposition 5.2.21);
- `tree_decomp(GraphWin& gw)`, creating a tree decomposition for all subgraphs generated by the graph separators and melting them into a global tree decomposition of the plane graph (see Theorem 5.2.23).

Concerning the dynamic programming phase, we implemented a class `TD` for tree decompositions with various subroutines, e.g.,

- `TD.make_nice()`, transforming a tree decomposition into a "nice tree decomposition" (see Lemma 5.1.3);
- `TD.reduce()`, reducing the size of the tree decomposition by combining two neighboring bags, whenever one appears to be a subset of the other.
- `TD.solve(problem Q)`, being the main procedures in the dynamic programming part for solving the currently chosen graph problem par-Q on the tree decomposition `TD.this`.

Although the LEDA package made many things much easier, the implementation work was fairly challenging, needing several new algorithmic ideas of problems not considered in the underlying theoretical papers. As memory quickly becomes a bottleneck, it was essential to encode the table entries of the dynamic programming as bit words, thus making it necessary to operate with "bit-masking" and dealing with bit parallelism at the word level. Another thing worth pursuing in order to save memory is to make a refined analysis of which tables have to be kept open (i.e., to be kept in main memory) during the dynamic programming process. In this context, it might be beneficial to experiment with choosing different root vertices of the tree decomposition. These and several more fine tunings were necessary to make the program competitive (see [5, 78]). Notably, all of these tunings are simple by themselves, but they are indispensable as a whole in order to get running times as gathered in the following.

Practical Challenges. Our implementation still has to be called a prototype—numerous future fine-tuning improvements are foreseeable. Examples for such improvements that we are about to incorporate are, e.g., given by

- experimenting with different planar embeddings (other than the straight-line embedding used so far) in order to further optimize Phase 1,
- making the construction of the tree decomposition more efficient (using heuristics, e.g., the ones proposed in [49, 131]) and also trying to further lower bag and thus table sizes,
- reducing the memory requirement for the tables in the dynamic programming, perhaps using ideas from [29] or [36], and also trying to bring the number of tables kept simultaneously in main memory at a minimum, and
- further easing the use of the software, e.g., by also providing meta-information such as expected remaining running time during the execution (so-called "progress indicators", see [38]).

6.1.3 Test Data

As test instances, we created a set of sample graphs using the LEDA [145] standard function

```
void random_planar_graph (graph& G, int n, int m)
```

for generating combinatorial random planar graphs. Here, n and m (with $m \leq 3n - 6$) specify the number of vertices and edges of the graph. The function, in a first step, generates a random maximal planar graph in an inductive way: For $n = 3$, as the induction base, a triangle is created. For $n > 3$, a random maximal planar graph of order $n - 1$ is generated, an additional vertex v is added to a random face f, and all edges from v to the boundary of f are drawn. In a second step, all but m edges are randomly removed. We remark that this method does not generate graphs according to the uniform distribution.

We created sample sets PGn (PG is short for "planar graphs") of random planar graphs with n vertices (where $n = 100$, 500, 750, 1000, 1500, 2000, 3000, 4000). Each sample set PGn contains 100 sample graphs. Here, for each graph in PGn, we chose m as a random number in the interval $[n - 1, 3n - 6]$. All graphs were given together with a "straight-line embedding" that was computed using the corresponding LEDA standard function [145].

Various graph structural data for the sample sets PGn is given in the first block of rows (see "Graph data") in Table 6.1. We measured for each graph G individually the following figures:

- *# vertices:* number of vertices in G;
- *# edges:* number of edges in G;
- *# layers:* number of layers of G using the straight-line embedding offered by LEDA;
- *max. degree:* maximum degree in G;
- *avg. degree:* average degree of G;
- *size of VC:* size of minimum vertex cover of G (as computed by our algorithm);
- *size of DS:* size of minimum dominating set of G (as computed by our algorithm);

Table 6.1 contains the average values over these figures for each graph sample set PGn.

We remark that we would have appreciated to also pursue intensive experiments on real-world instances. But, as a matter of fact,—despite numerous efforts—it turned out to be extremely difficult to obtain meaningful, non-commercial, and publicly available real-world data, such as, e.g., models of transportation networks or models of socio-economic networks. As an alternative set of (randomly generated) test instances we worked with instances from an Internet topology generators (see [121]), which creates random network topologies based on the so-called "Autonomous System" connectivity in the Internet. In general, we might say that these graphs, however, turned out to be less challenging than the sample sets PGn. This is why we concentrate on the sample sets PGn in what follows.

The underlying machine for all tests in Sections 6.2, 6.3, and 6.4 is a conventional LINUX PC with 750MHz Pentium III processor and 720 MB main memory.

Tree Decomposition Based Algorithms

sample set	PG100	PG500	PG750	PG1000	PG1500	PG2000	PG3000	PG4000
Graph data:								
# vertices	100	500	750	1000	1500	2000	3000	4000
# edges	201.5	974.6	1483.7	1978.9	2992.0	3960.8	6070.6	8264.5
# layers	3.92	5.12	5.36	5.61	5.84	6.11	6.29	6.86
max. degree	23.2	50.8	61.2	73.3	90.6	104.9	129.6	146.6
avg. degree	4.03	3.90	3.96	3.96	3.99	3.96	4.05	4.13
size of VC	47.2	225.2	342.0	453.9	683.6	917.3	1373.8	1856.8
size of DS	24.2	126.4	183.6	247.3	360.9	472.9	719.1	989.4
Tree decompositions obtained (by Phase 1):								
width of tree dec.	6.99	9.33	10.32	11.11	12.24	12.85	13.91	15.31
highest occurring width	11	15	14	19	20	20	22	21
avg. bagsize	4.20	4.22	4.33	4.37	4.46	4.57	4.62	4.72
# tree nodes	75.7	389.9	583.2	779.6	1167.2	1552.3	2327.7	3087.1
depth of tree	19.1	52.8	70.6	82.5	115.2	122.6	163.0	196.3
max. degree in tree	7.0	29.1	37.0	54.3	66.3	89.3	150.1	185.7
Time needed:								
time (sec): Phase 1	0.34	2.96	6.90	12.46	30.20	48.73	128.38	253.05
time (sec): Phase 2	0.06	5.41	12.35	34.46	116.40	192.17	402.36	1015.91
total time (sec)	0.40	8.37	19.25	46.92	146.60	240.90	530.74	1268.96

Table 6.1: Summary of experimental results for tree decomposition based algorithms. The table shows various data on the structure of the tree decompositions generated by our algorithm. The numbers in the various rows are taken as the *averages* over 100 graphs in PGn of the corresponding column.

6.2 Evaluation of Tree Decomposition Based Algorithms

In this subsection, we report on an empirical evaluation of the tree decomposition based algorithms provided by the "FPT-Toolbox" described in Section 6.1. In particular, we want to gain insight on the structure of the tree decompositions generated by our algorithms. As test data we used the samples PGn of combinatorial random planar graphs that were generated as described in Subsection 6.1.3. To keep the discussion within reasonable length, we restrict ourselves to the par-VERTEX COVER problem on planar graphs. At the time being, the implementation concerning this problem is the one that achieved the highest level of maturity. We ran the combination of both, Phase 1 (i.e., the construction of a tree decomposition of the given plane graph) and Phase 2 (i.e., the dynamic programming on the obtained tree decompositions) in order to solve par-VERTEX COVER. In the sequel, we partly follow [5].

Settings Recall the general outline of the algorithm (see Subsection 6.1.1). The tradeoff-parameter ψ allows us to tune this algorithm in two opposing directions. For a large value of ψ, the algorithm allows for large separators, but only graph chunks with few layers. Conversely, for a small value of ψ the algorithm only uses small separators, but admits graph chunks with many layers. In the extremal case (i.e., when the tradeoff-parameter ψ is set to zero), no separation is done and the graph is considered as a single "big" chunk which is processed by the algorithm in Theorem 5.2.4. In this extremal case, we get

$$\mathrm{tw}(\mathcal{X}) \leq 3 \cdot \mathrm{out}(G, \phi) - 1, \tag{6.1}$$

where $\mathcal{X}$ is the tree decomposition obtained for the plane graph (G, ϕ) (see Theorem 5.2.4).

From a theoretical point of view, the optimal tradeoff-parameter is given by $\psi_{\mathrm{opt}} := \sqrt{3/2}$ (see the proof of Theorem 5.2.23). Here, "optimal" means that the worst-case upper bound on the width of the resulting tree decomposition $\mathcal{X}$ is smallest possible, namely

$$\mathrm{tw}(\mathcal{X}) \leq 2\sqrt{6dk} + (3w - 1), \tag{6.2}$$

where d and w are the size-factor and the width of the underlying LSP-problem (see Theorem 5.2.23). One can construct examples (see Remark 5.2.18), where the parameter k (i.e., the size of an optimal vertex cover) is linearly related to $\mathrm{out}(G, \phi)$. In this sense, the upper bound given in Eq. (6.2) is (asymptotically) superior to the one in Eq. (6.1)

From a practical point of view, the situation seems different: It turned out that, on the one hand, the graphs generated in our setting had few layers only using a simple straight-line embedding ϕ, i.e., the number $\mathrm{out}(G, \phi)$ is small. In nearly all cases we had less than 10 layers. On the other hand, the size of an optimal vertex cover or an optimal dominating set, i.e., the parameter value for k seemed to be large: As a rule of thumb, for the graphs generated in our setting, we may say that an optimal vertex cover contained about half of the vertices of the graph. Hence, for our combinatorial random graphs, the upper bound in Eq. (6.1) yields smaller values than the upper bound from Eq. (6.2). As a consequence, trading the parameter k for the parameter $\mathrm{out}(G, \phi)$ seems reasonable in our setting. In deed, our experiments showed that the widths of the tree decompositions obtained without doing separation were much smaller compared to the ones obtained after separation. Hence, for all tests, the tradeoff-parameter ψ was set to zero.

Evaluation. In this setting, we measured the following figures for each given random input graph G individually:

- *width of tree dec.:* width of tree decomposition $\mathcal{X}$ obtained for G (by Phase 1);
- *avg. bagsize:* average size of the bags of the tree decomposition $\mathcal{X}$;
- *# tree nodes:* number of tree nodes of tree decomposition $\mathcal{X}$;
- *depth of tree:* depth of the tree in tree decomposition $\mathcal{X}$;
- *max. degree in tree:* maximum degree of the tree in tree decomposition $\mathcal{X}$;

The averages of these values over all graphs from a given sample set are summarized in the second block of rows (see "Tree decompositions obtained") in Table 6.1. Moreover, the table shows, the highest width of a tree decomposition that occurred in a given sample set.

Besides, we recorded the running time for the two phases. The corresponding values can be seen in the third block of rows in Table 6.1 (see "Time needed").

In addition, for each input graph, we investigated the distribution of the various sizes of different bags that appeared in the tree decomposition. More precisely, for each graph G and

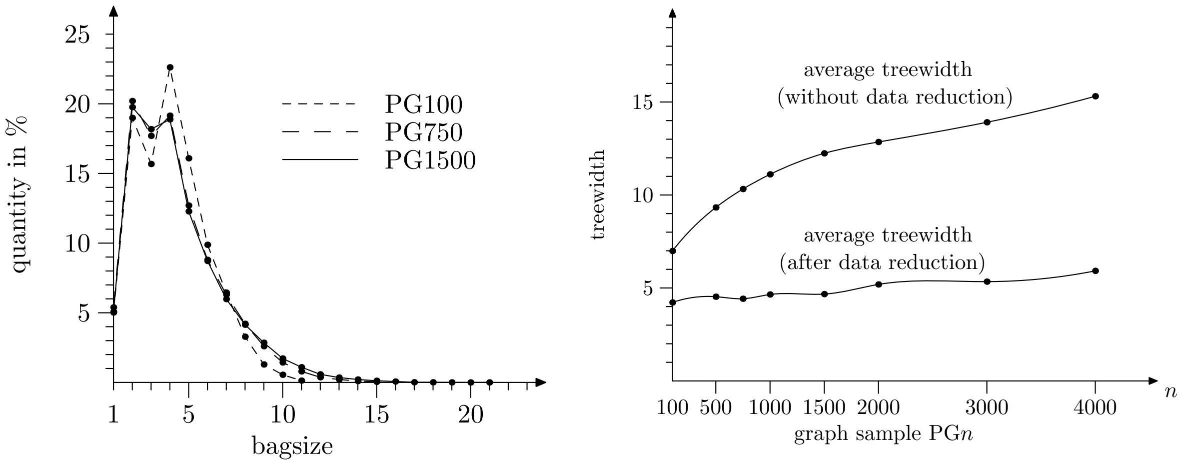

Figure 6.4: Bag Distribution and Treewidth. The left diagram illustrates the distribution of sizes of bags in tree decompositions obtained by Phase 1 (without data reduction). The right diagram compares the treewidth values obtained without running preprocessing Phase 0 to the values obtained after the data reduction Phase 0.

the tree decomposition $\mathcal{X}$ obtained, we explored the percentage of bags having size s (where $s \in [1, tw(\mathcal{X}) + 1]$). This distribution is fundamental for the running time of Phase 2 of the algorithm. The left-hand diagram in Fig. 6.4 shows this distribution averaged over the graphs of selected sample sets PGn.

Discussion. Our main observation concerning Phase 1 of the algorithm is that the widths of the tree decompositions obtained in practice (see row "treewidth" in Table 6.1) are much lower than predicted from the theoretical bounds. The upper bound from Eq. (6.1) and, in particular, the one from Eq. (6.2) are much too pessimistic. Take, e.g., the sample set PG750 where the average treewidth is 10.3. Observing that the average number of layers of these graphs is 5.36, and using this value in the worst-case upper bound from Eq. (6.1), we expect treewidth around $3 \cdot 5.36 - 1 \approx 15$. The actual value of 10.3 obtained for our combinatorial random graphs is by a factor 1.46 lower than this value. The same holds true for the graph in PG750 which has the highest treewidth of 14. This graph, however, has 7 layers which means that the worst-case upper bound would have guaranteed a width of 20 (by a a factor 1.43 worse than what we obtained). Note that the average size of a minimum vertex cover for the sample PG750 is around 342 (see row "size of VC" in Table 6.1). For such a value, the theoretical worst-case upper bound in Eq. (6.2) would yield $4\sqrt{3 \cdot 342} + 5 \approx 133$, which is by a factor 12.9 (sic!) higher than what we achieved in practice.

Moreover, we made two further interesting observations:

- The average size of the bags in the tree decompositions in *all* of the samples PGn is very small, namely around 4.5 (seemingly independent of the width of the tree decompositions and the size of the input graphs). To illustrate this, consider the left-hand diagram of Fig. 6.4, which shows the distribution of the sizes of the bags for various sample sets PGn.

Note that the size of a very high percentage of the bags is in the range of $\{1, \ldots, 6\}$, and only few bags are large (determining the width of the tree decomposition).

- The number of nodes in the tree decomposition is lower than the guarantee given in Theorem 5.2.4. There, we would expect $2n - 1$ tree nodes. The tree decompositions in average turn out to have only around $0.75n$ many bags. This improvement by a factor of 2.7 over the expected number of bags is due to an implemented heuristic which reduces the size of the tree decomposition by combining two neighboring bags, whenever one appears to be a subset of the other.

Both, the distribution of the bagsizes and the number of bags have a direct influence on the running time of Phase 2 of the algorithm: Very recently, the notion of f-*cost* was introduced as a more refined measure for the quality of a tree decomposition [47]. Here, $f : \mathbb{N} \to \mathbb{R}^+$ is some function and the f-cost of a tree decomposition $\mathcal{X} = \langle \{X_i \mid i \in I\}, T \rangle$ is defined to be $\Sigma_{i \in I} f(|X_i|)$. Since, in Phase 2, the time (and space) needed to process a node of the tree decomposition whose associated bag has size k roughly is $f(k) = 2^k$, the time needed for Phase 2 is $f(\mathcal{X}) := \Sigma_{i \in I} f(|X_i|)$. In this sense, the distribution of the bag sizes (and the low average bag size) measured in our experiments contribute to a small f-cost and, hence, a fast running time for Phase 2.

Note that we found vertex covers of average size 1370 in PG3000 in less than ten minutes average time. This might be compared with recent experimental results of Dehne *et al.* [69], where search tree algorithms for VERTEX COVER on *general* graphs were parallelized on 10 Sun SPARC workstations. Within a similar time range, they optimally solved VERTEX COVER instances for parameter values k only around 400.

We only briefly mention that we carried out similar tests for par-DOMINATING SET. Recall that for this problem Phase 2 (see Section 5.3.2) is more time and space consuming. Thus, the width and the structure of the tree decomposition obtained by Phase 1 play a more important role here. In addition, since the the dynamic programming presented in Section 5.3.2 relies on a nice tree decomposition, the original decomposition needs to be transformed generating considerably more tree nodes. On the positive side, our observations revealed that we can proceed Phase 2 for graphs from the sample set PG1000, in case the width of the nice tree decomposition is around 10, in an average of around five minutes. On the negative side, the limits in terms of memory requirements were exceeded for higher width. Concerning space efficiency, it is an open challenge to incorporate and explore further ideas such as, e.g., the ones from [29] or [36].

Summary. To put it in a nutshell, the key message from the experimental studies in this subsection is that—at least on the random planar graphs used in our setting—Phases 1 and 2 perform much better than could have been expected by their worst-case analysis in Chapter 5. The low total running time for solving par-VERTEX COVER on these instances is mainly due to the relatively well-behaving tree decompositions (both in terms of width and structure). The pure worst-case upper bounds derived from our theoretical analysis seems much too pessimistic for practical purposes. We hereby reinforced grounds for the practical significance of the concept of tree decompositions of graphs.

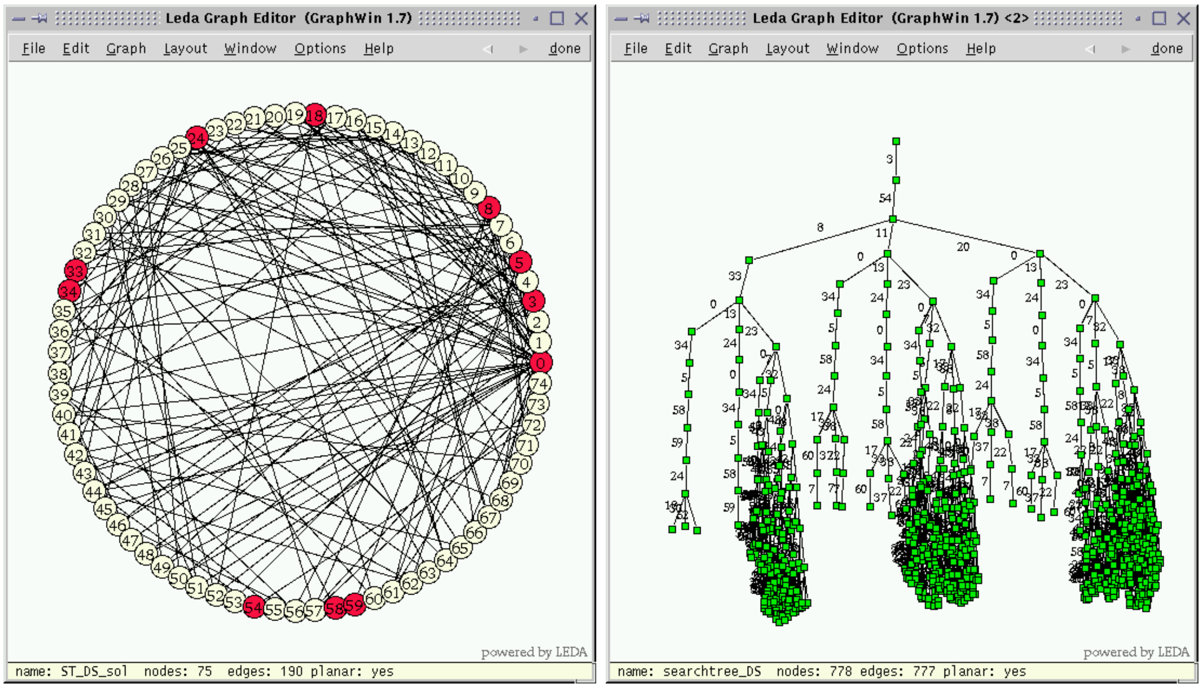

Figure 6.5: Example for the search tree algorithm for par-DOMINATING SET. The left-hand diagram shows a (planar) graph drawn in a circular layout for which an optimal dominating set (red vertices) was computed by our degree-branching search tree algorithm. The right-hand diagram illustrates the corresponding search tree. It has depth 11 and contains 778 search tree nodes. The labels on the edges correspond to the vertices of the original graph that were chosen to belong to the dominating set in the specific branch.

6.3 Evaluation of a Search Tree Based Algorithm

Accompanying the realization of the "FPT-Toolbox" which mainly uses the tree decomposition based algorithms of Chapter 5 for various LSP-problems, the search tree algorithm for par-DOMINATING SET from Chapter 3 was implemented by Simone Lehnert [135] in a student project under my supervision. We now very briefly report on a test series for this algorithm.

An example of the computation of an optimal dominating set using our search tree method is given in Fig. 6.5.

Recall the search tree algorithm for par-(ANNOTATED) DOMINATING SET from Section 3.2: It is based on the degree-branching method, i.e., in each node of the search tree we determine a (black) "branching vertex" v of lowest degree. Then, we branch according to this low-degree vertex, i.e., we take v or one of its neighbors in the dominating set we seek for. The neighbors of the vertices that are already determined to belong to the dominating set are marked (white). White vertices can be assumed to be dominated but still are candidates for the dominating to be constructed. To guarantee the existence of a *black* vertex of degree at most seven (and, thus, a bounded branching degree), in each search tree node, the corresponding graph has to be reduced

Search Tree Algorithm

sample set	PG25	PG50	PG75	PG100
Graph data:				
# vertices	25	50	75	100
# edges	47.0	98.0	148.1	201.5
max. degree	10.7	15.9	20.0	23.2
avg. degree	3.76	3.92	3.95	4.03
Search tree obtained:				
size of search tree	28.7	307.9	3447.2†	15808.3‡
maximum branching number	2.36	2.61	2.79†	2.99‡
average branching number	1.62	1.57	1.56†	1.62‡
depth of search tree	7.6	13.9	20.2†	24.0‡
time (sec)	0.07	3.54	172.44	948.5‡
Application of bw-Rules per search tree node:				
bw-Rule 1	9.8	22.2	33.9	46.4
bw-Rule 2	8.8	18.7	29.9	38.2
bw-Rule 3	0.7	1.0	1.1	1.4
bw-Rule 4	0.002	0.001	0.004	0.001

† 3 test runs were interrupted after 50000 search tree nodes.
‡ 25 test runs were interrupted after 50000 search tree nodes.

Table 6.2: Summary of experimental results for the search tree based algorithm for par-DOMINATING SET. The numbers in the various rows are taken as the *average* over graphs in PGn of the corresponding column.

with respect to bw-Rules 1, 2, 3, and 4 (see Subsection 3.2.2).

Evaluation. The tests were performed on combinatorial planar graph samples PG25, PG50, PG75 and PG100 (of 100 graphs each) that were created as described in Subsection 6.1.3.[2] For each graph in a given sample set PGn, a search tree was built as described in Subsection 3.2.1 (also refer to the general scheme in Fig.3.1)—using the branching rule (3.7). In addition, we incorporated a trivial branch and bound strategy for degree-one branching vertices.[3] Recall that the *branching number* of a specific search tree node corresponds to the number of children generated at this search tree node. This branching number is given by $\deg(v) + 1$, where v is the corresponding branching vertex used in this search tree node.[4]

Our main motivation was to get insight into the branching numbers that are used through the algorithm. For this reason, we did not limit the height of the search tree a priori by some parameter k, but we computed the search tree (in a breadth first manner) until an optimal dominating set was found. To keep the running time in reasonable time, we aborted the algorithm after a limit of 50000 search tree nodes.

For each graph, we measured the following figures.

[2]Compared to Section 6.2, we had to use smaller test instances since otherwise the search trees would have been too large.

[3]If the branching vertex v has degree one, then, the branch that would (formally) be created for v is discarded, since it is always optimal to take the neighbor of v into the dominating set.

[4]Since we do not branch if $\deg(v) = 1$, the branching vector in such a situation is 1 instead of 2.

- *size of search tree:* number of search tree nodes needed to find an optimal dominating set;
- *maximum branching number:* maximum branching number that occurred in the search tree, i.e., the maximum number of children of a search tree node;
- *average branching number:* average branching number that occurred in the search tree, i.e., the average number of children of a search tree node;
- *depth of search tree:* depth of the search tree which corresponds to the size of an optimal dominating set if the search tree node limit has not yet been reached;

In addition, we recorded the time (in seconds) that was needed to construct the search tree. Moreover, we measured how often the bw-Rules 1, 2, 3, and 4 were applied per search tree node in the corresponding reduction step.

The averages of these values over all graphs from a given sample are summarized in in Table 6.2.

Discussion. It is interesting to see that the search trees generated in our setting had considerably fewer nodes than could have been expected from the worst-case bound. According to the worst-case scenario, for a search tree of height k, we can have

$$\sum_{i=0}^{k} 8^i = \frac{1}{7}(8^{k+1} - 1) \tag{6.3}$$

many search tree nodes, because the theoretical upper bound on the branching number is 8. Consider the sample set PG25. Observing that the average depth of a search tree here is 7.6 and plugging this value in the Formula (6.3) of the worst-case analysis yields $8.35 \cdot 10^6$ search tree nodes. Compare this number with the average of 28.7 search tree nodes obtained for PG25. The situation is more drastic in the case of PG100: Using the average depth of 24.0, the worst-case analysis in (6.3) yields $5.40 \cdot 10^{21}$ search tree nodes. Assuming the same type of machine we used (with 10^9 operations per second) and assuming that each search tree node could be processed by a single operation, constructing such a search tree would theoretically take around 228000 years. Instead, we measured approximately 950 seconds average time in our tests.

This huge gap between worst-case analysis and average-case behavior on our test sets can be explained by the branching numbers measured in the tests. Recall that the worst-case upper bound on the branching number is tight. In Fig. 3.4 we presented a generic example of a reduced graph for which all black vertices have degree 7, resulting in a branching vector of 8. However, such a worst-case scenario never appeared in our combinatorial random graphs. Moreover, the worst-case branching vector that occurred over all graphs in the sample set is 4 (sic!). In average, the branching number is as low as 1.6, seemingly independent of the size of the graphs considered. The average of the maximum branching number that occurred within a sample set ranges from 2.36 (for PG25) to 2.99 (for PG100). The moderate running times can be explained by the surprisingly low values for the branching numbers.

Still, the practicability of this search tree algorithm has its limits. The running time grows exponentially with the size of the dominating set. Despite of a low average branching number, we were not able to compute dominating sets for the sample set PG500 (from Section 6.2) within a few minutes range, whereas—due to the low outerplanarity number of these graphs—this was easily possible using the tree decomposition based approach.

Finally, we observe that, in the reduction step, bw-Rules 1 and 2 were applied more frequently than the slightly more involved bw-Rules 3 and 4. From a practical point of view, it might be reasonable to omit the latter rules, since one of the most time-consuming steps in each search tree node is to detect situations in which bw-Rules 3 and 4 can be applied. Then, however, a low-degree black branching vertex cannot be guaranteed from a theoretical point of view, and we lose the guaranteed upper bound on the running time.

Summary. Though the worst-case upper bound on the branching number for the search tree algorithm for par-DOMINATING SET is tight, it seems to be much too pessimistic for the test graphs used in our setting. The algorithm appears to be practical for moderate parameter values and, hence, for exactly solving (ANNOTATED) DOMINATING SET on graphs of moderate sizes of an optimal dominating set.

6.4 The Influence of Data Reduction by Preprocessing

We now focus on analyzing the potential of data reduction by preprocessing. On the one hand, we investigate the strength of the well-known problem kernelization for par-VERTEX COVER attributed to Nemhauser and Trotter (see Theorem 2.1.7 and Corollary 2.1.8); the theoretical upper bound on the size of the problem kernel, here, being $2\,\mathrm{vc}(G)$. On the other hand, our interest lies in the practical usefulness of our linear problem kernelization for par-DOMINATING SET on planar graphs (see Theorem 2.2.1); the theoretical upper bound on the problem kernel, in this case, being $335\,\mathrm{ds}(G)$. Our findings in this section reveal that, for the combinatorial random graph test instances, the reduction rules for par-DOMINATING SET seem to be more powerful than the problem kernel reduction due to Nemhauser and Trotter.

6.4.1 Nemhauser-Trotter Kernelization for VERTEX COVER

A similar series of experiments as described in Section 6.2 was carried out with an additional preprocessing Phase 0 to perform data reduction. We again used the sample sets PGn of combinatorial random graphs (see Subsection 6.1.3) to evaluate the power of the problem kernel reduction due to Nemhauser and Trotter (see Theorem 2.1.7). It is important to note that by applying this preprocessing we are no longer able to generate *all* minimum vertex covers.

Evaluation. For each graph in a set PGn, we iteratively applied the data reduction due to Nemhauser and Trotter[5] until the graph could not be reduced any further. Recall that the construction of Nemhauser and Trotter (see Theorem 2.1.7) computes on input $G = (V, E)$ two disjoint subsets of the vertices: a set C_0 that can be assumed to belong to an optimal vertex cover, and a set V_0 of possible further candidates for an optimal vertex cover. All other vertices, i.e., $V \setminus (C_0 \cup V_0)$, can be removed from the graph. We measured the following figures:

[5] In fact, we slightly heuristically tuned the data reduction algorithm suggested by Nemhauser and Trotter.

Data Reduction for Vertex Cover

sample set	PG100	PG500	PG750	PG1000	PG1500	PG2000	PG3000	PG4000
Preprocessing Nemhauser-Trotter:								
# vertices removed	56.2	337.9	518.1	684.2	1054.9	1347.1	1963.9	2507.1
(percentage)	56.2%	67.6%	69.1%	68.4%	70.3%	67.4%	65.5%	62.7%
# edges removed	103.7	642.7	1045.0	1372.3	2166.2	2660.8	3999.3	5467.0
(percentage)	51.5%	65.9%	70.4%	69.3%	72.4%	67.2%	65.9%	66.1%
# vertices for VC found	23.1	133.8	209.7	273.9	429.3	544.9	783.8	1007.1
(percentage) †	48.9%	59.4%	61.3%	60.3%	62.8%	59.4%	57.1%	54.2%
time (sec)	0.30	1.38	2.63	4.06	8.60	12.24	30.90	60.45

† percentage with respect to an optimal vertex cover.

Table 6.3: Summary of experimental results for the data reduction due to Nemhauser and Trotter. All values in a row are taken as the *average* over graphs in PGn of the corresponding column.

Data Reduction + Tree Decomposition Based Algorithm for Vertex Cover

sample set	PG100	PG500	PG750	PG1000	PG1500	PG2000	PG3000	PG4000
Tree decompositions obtained (for remaining reduced instances):								
width of tree dec.	4.22	4.53	4.42	4.65	4.67	5.19	5.34	5.92
highest occurring width	10	9	10	12	16	14	14	16
avg. bagsize	3.70	3.50	3.43	3.41	3.37	3.48	3.46	3.54
# tree nodes	27.7	97.9	136.8	188.9	287.8	399.8	626.5	912.3
depth of tree	8.1	15.6	15.3	17.1	17.6	25.0	26.3	33.8
max. degree in tree	3.2	11.9	19.7	24.7	39.9	49.4	71.9	94.3
Time needed:								
time (sec): Phase 0	0.30	1.38	2.63	4.06	8.60	12.24	30.90	60.45
time (sec): Phase I	0.15	0.63	1.02	1.52	3.97	7.54	20.51	49.15
time (sec): Phase II	0.08	0.20	0.21	0.17	0.22	0.16	0.13	0.24
total time (sec)	0.53	2.21	3.86	5.75	12.79	19.93	51.54	109.84

Table 6.4: Summary of experimental results for the combination of data reduction and tree decomposition based algorithms. The table shows various data on the structure of the tree decompositions of the reduced instances. In addition, the running times measured for Phase 0 (data reduction), Phase 1 (construction of tree decompositions), and Phase 2 (dynamic programming on tree decomposition) are recorded. The numbers in the various rows are taken as the *average* over graphs in PGn of the corresponding column. Compare these results with the ones obtained without the data reduction (see Table 6.1).

- *# vertices removed:* the number of vertices removed by the data reduction;
- *# edges removed:* the number of edges removed by the data reduction;
- *# vertices for VC found:* the number of vertices that could be determined by the preprocessing to be in an optimal vertex cover (i.e., the size of C_0);

Besides, we recorded the time needed to perform this data reduction. Table 6.3 summarizes these values averaged over each sample set PGn individually.

After the data reduction (Phase 0), we are left with the reduced, computationally challenging

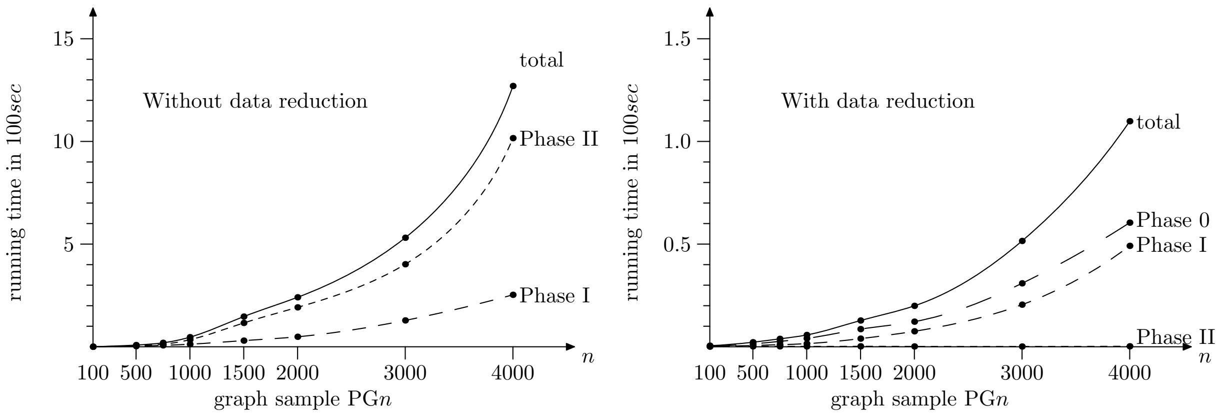

Figure 6.6: Running time. The left-hand and the right-hand diagram, respectively, show (the various contributions for) the running times without the preprocessing (see Section 6.2) and with the preprocessing (see Subsection 6.4.1), respectively. Note that the time-axis is scaled down by a factor of 10 in the right-hand diagram.

graphs from our sample set PGn. On these reduced instances we ran Phases 1 and 2 of the tree decomposition based algorithms, while—analogously to the tests in Section 6.2—recording the relevant figures for the structure of the tree decompositions. Table 6.4 gives an overview on the structure of the tree decompositions that were computed for the reduced graphs.

Finally, the running times for the various phases are collected in the second block of rows in Table 6.4.

Discussion. The data reduction has an impressive impact on both the size of the remaining reduced graphs and the width of their tree decompositions: In average,

- around 67% of the vertices, and
- around 66% of the edges of the original graphs

were removed by the preprocessing phase (see Table 6.3). Moreover, the preprocessing detected a very high percentage (in average, around 58%) of vertices that can be guaranteed to belong to an optimal vertex cover.

The width of the tree decompositions for the reduced problem kernels (see Table 6.4) are considerably smaller than the width of the tree decompositions obtained for the original graphs (see Table 6.1). As an example, again take the (reduced) graphs from PG750, where the width of the computed tree decompositions in average now is 4.42, whereas the average width for the original graphs was 10.32, a decrease by more than 57%. We refer to Fig. 6.4 for a comparison of the average treewidths obtained with and without data reduction, respectively. The number of layers decreased from an average of 5.36 to 2.57. In all sample sets (seemingly independent of the size of the original graphs) we observed an average bagsize of around 3.5 for the reduced graphs (compared to 4.5 for the original graphs).

As a result of the well-behaving tree decompositions for the reduced graph instances, we obtain a drastic improvement of the running times for Phase 1 and, especially, for Phase 2.

Whereas in the setting of Section 6.2, Phase 2 played a crucial role in the overall running time, the contribution of this phase is almost negligible when running the preprocessing (see Fig. 6.6 for an illustration). The major part of the overall running time now is given by the preprocessing Phase 0 itself.

The key message is that, considering the overall running time of the algorithm, data reduction pays off. The larger the graphs are, the better the speed-up gained by the preprocessing: Whereas we get an average speed-up by a factor of approximately 3.8 for smaller graphs from PG500, the corresponding factor is around 11.6 for the graphs in PG4000.

Summary. The tests in this section revealed that—at least on the random planar graphs used here—the data reduction suggested by Nemhauser and Trotter allows significant improvement concerning the width of the tree decompositions and, thus, drastically speeds-up the overall running time for the whole algorithm. The only negative point herein is that no more all optimal vertex covers can be generated.

6.4.2 Kernelization for Dominating Set

Finally, we investigated the strength of the problem kernelization for par-DOMINATING SET on planar graph (see Section 2.2). Recall that the problem kernelization was based on two simple reduction rules (see Subsection 2.2.1) that are repeatedly applied to an input graph.

We remark that, in our experiments, we used a slight modification of the reduction rules: Formally, when Rule 1 or Rule 2 (see Subsection 2.2.1) is applied and some vertex v is determined to belong to an optimal dominating, the reduction rules attach a gadget vertex v' of degree one to v. In our setting here, we simply removed the vertex v from the graph and "marked" its neighbors as being already dominated. In this sense, we dealt with the variant of par-DOMINATING SET called par-ANNOTATED DOMINATING SET (see Definition 3.2.2), where the input instances are black and white graphs consisting of two types of vertices: black vertices which still need to be dominated; and white vertices which are assumed to be already dominated. A slight modification makes Rule 1 and Rule 2 applicable to such instances as well.

These two reduction rules then were enriched by the four very simple bw-Rules for par-ANNOTATED DOMINATING SET that were developed in the search tree construction (see Subsection 3.2.2 for details).

An example of a problem kernel reduction as performed by our software package is given in Fig. 6.7.

Evaluation. The potential of the aforementioned reduction rules was tested separately. We ran a series of tests using Rule 1 only, using Rule 2 only, using a combination of Rule 1 and Rule 2, and, finally, using Rules 1 and Rule 2 together with the four bw-Rules.

For each test run, we measured the following figures:

- *# vertices removed:* the number of vertices removed by the data reduction;
- *# edges removed:* the number of edges removed by the data reduction;

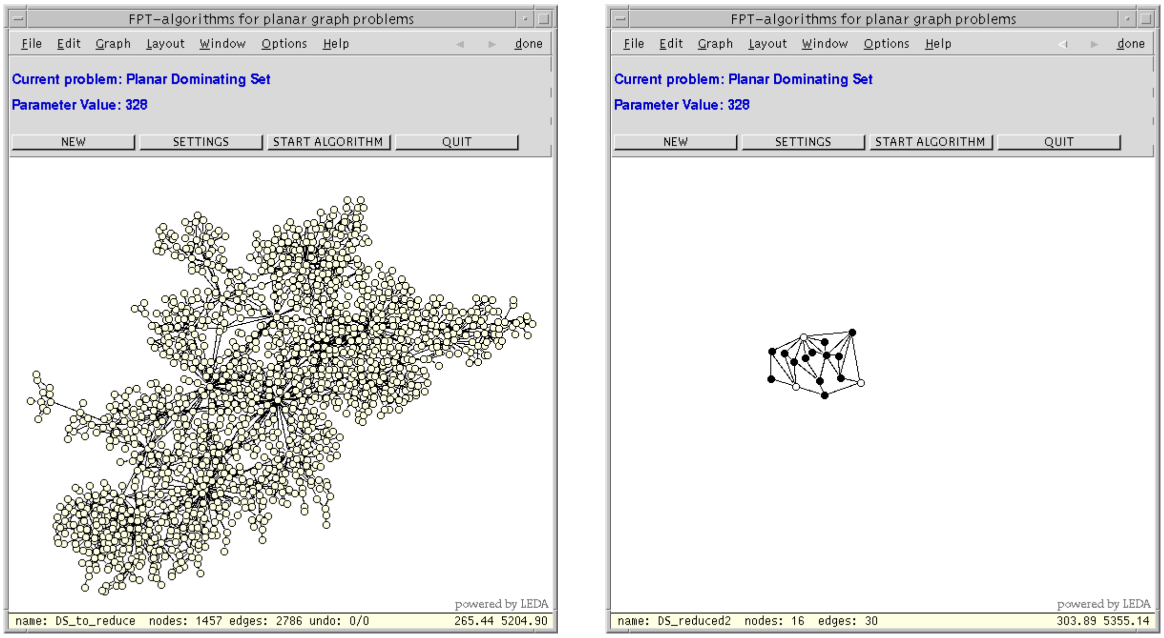

Figure 6.7: Example for the power of our data reduction through preprocessing. The planar graph (with around 1500 vertices) in the left-hand side window is reduced to a black and white graph instance with 16 vertices (shown in the right-hand side window).

- *# vertices for DS found:* the number of vertices that could be determined by the preprocessing to be in an optimal dominating set;

In addition, the time needed in order to reduce the graph with respect to the given set of rules was recorded.

The results of the tests are summarized in Table 6.5.

Discussion. The preprocessing seems to be very effective—at least on the given random sample sets. Using the combination of reduction Rules 1 and 2, as they were used to proof the linear problem kernel, we may say that, in average,

- more than 79% of the vertices and
- more than 88% of the edges

were removed from the graph. Moreover, the reduction rules determined a very high percentage (in average, more than 89%) of the vertices of an optimal dominating set—seemingly independent of the size of the input graphs. The overall running time for the reduction ranged from less than one second (for small graph instances with 100 vertices) to around 30 seconds (for larger graph instances with 4000 vertices).

Data Reduction for Dominating Set

Rule 1:

sample set	PG100	PG500	PG750	PG1000	PG1500	PG2000	PG3000	PG4000
# vertices removed	68.0	358.3	528.3	716.8	1058.8	1431.6	2183.0	2887.6
(percentage)	68.0%	71.7%	70.4%	71.7%	70.6%	71.6%	72.8%	72.2%
# edges removed	151.6	771.8	1174.6	1577.9	2388.5	3164.2	4918.4	6774.1
(percentage)	75.2%	79.2%	79.2%	79.7%	79.8%	79.9%	81.0%	82.0%
# vertices for DS found	19.3	104.2	149.3	203.2	292.1	388.9	594.9	826.8
(percentage)†	79.9%	82.4%	81.1%	82.2%	80.9%	82.2%	82.7%	83.6%
time (sec)	0.22	0.91	3.48	6.36	3.51	4.76	10.33	14.69

Rule 2:

sample set	PG100	PG500	PG750	PG1000	PG1500	PG2000	PG3000	PG4000
# vertices removed	73.9	358.4	530.8	711.8	1054.0	1415.6	2141.9	2833.1
(percentage)	73.9%	71.7%	70.8%	71.2%	70.3%	70.8%	71.4%	70.8%
# edges removed	176.3	837.0	1265.3	1696.3	2553.9	3386.8	5232.3	7157.8
(percentage)	87.5%	85.9%	85.3%	85.7%	85.4%	85.5%	86.2%	86.6%
# vertices for DS found	18.8	95.4	137.7	185.7	267.1	354.3	540.2	744.8
(percentage)†	78.0%	75.5%	75.0%	75.1%	74.0%	74.9%	75.1%	75.3%
time (sec)	0.26	1.28	4.97	7.73	9.27	8.45	23.31	35.94

Rule 1+2:

sample set	PG100	PG500	PG750	PG1000	PG1500	PG2000	PG3000	PG4000
# vertices removed	78.6	398.4	585.9	793.5	1167.8	1585.1	2397.2	3160.7
(percentage)	78.6%	79.7%	78.1%	79.4%	77.9%	79.3%	80.0%	79.0%
# edges removed	178.3	866.4	1305.2	1755.7	2634.5	3515.5	5427.5	7386.9
(percentage)	88.5%	88.9%	88.0%	88.7%	88.1%	88.8%	89.4%	89.4%
# vertices for DS found	21.5	113.1	162.5	220.8	316.9	423.3	640.2	890.2
(percentage)†	89.2%	89.5%	88.5%	89.3%	87.8%	89.5%	89.0%	90.0%
time (sec)	0.58	1.86	10.19	14.55	16.07	10.21	27.35	33.90

Rule 1+2 + bw-Rules 1, 2, 3, 4:

sample set	PG100	PG500	PG750	PG1000	PG1500	PG2000	PG3000	PG4000
# vertices removed	99.9	498.2	747.1	997.4	1496.2	1994.5	2992.3	3987.1
(percentage)	99.9%	99.6%	99.6%	99.7%	99.7%	99.7%	99.7%	99.7%
# edges removed	201.4	971.9	1478.9	1974.9	2986.7	3962.0	6059.2	8245.7
(percentage)	99.9%	99.7%	99.7%	99.8%	99.8%	99.8%	99.8%	99.8%
# vertices for DS found	24.1	125.9	182.8	246.5	359.4	471.1	717.5	985.5
(percentage)†	99.8%	99.6%	99.6%	99.7%	99.6%	99.6%	99.8%	99.6%
time (sec)	0.01	0.43	0.94	2.35	4.14	4.80	6.91	9.50

† percentage with respect to an optimal dominating set.

Table 6.5: Summary of experimental results for the data reduction of par-dominating set instances. The numbers in the various rows are taken as the *average* over graphs in PGn of the corresponding column.

Surprisingly, using Rule 1 or Rule 2 *alone* already resulted in a very powerful data reduction. In both cases, in average, more than 71% of the vertices could be removed from the graph. Clearly, reducing a graph with respect to Rule 1 is less time-consuming than reducing a graph with respect to the more complex Rule 2. Moreover, Rule 1 seemed to be stronger than Rule 2 in the detection of vertices of an optimal dominating (in average, 81% compared to 75%). Conversely, we noticed a subtle tendency that Rule 2 removed more edges compared to Rule 1 (in average, 86% compared to 80%).

Finally, enriching Rules 1 and 2 with the four simple bw-Rules led to an extremely powerful data reduction on our set of random instances. Most interestingly, the combination of these rules removed, in average,

- more than 99.7% of the vertices and
- more than 99.8% of the edges

of the original graph. More than 99.6% of the vertices that belong to an optimal dominating set could be detected. These percentages again seem to be independent of the size of the input graph. We observed that in this extended setting, the running times for the data reduction went down to less than half a second (for graphs of 100 vertices) and less than ten seconds (for graphs of 4000 vertices) in average. This is due to the fact that we tried to apply simple reduction rules (such as the easy bw-Rules) with a higher priority than more complicated rules (such as Rule 2). Hence, the time-consuming sophisticated steps had to be carried out on small graphs only.

Finally, let us compare our data reduction for par-DOMINATING SET with the data reduction for par-VERTEX COVER attributed to Nemhauser and Trotter (see Section 6.4.1). In practice, the data reduction for par-VERTEX COVER reduced a considerably smaller percentage of the vertices than our data reduction for par-DOMINATING SET. This, however, could not have been expected from the corresponding worst case analysis: recall that Nemhauser and Trotter yields a problem kernel of size $2k$, whereas for our data reduction a problem kernel size of $335k$ was shown.

Summary. Our experimental studies underpin the big potential of the presented reduction rules for par-DOMINATING SET, leading to graph size reductions of more than 99% when experimenting with random graphs. An optimal par-DOMINATING SET for the small remaining reduced graphs can easily be computed using either a tree decomposition based algorithm (see Section 6.2) or a search tree based algorithm (see Section 6.3). Hence, we anticipate that every future algorithm for DOMINATING SET, whether approximation, fixed-parameter, or purely heuristic, always should take into consideration the data reduction method proposed here.

Chapter 7

Conclusion

The aim of this work was to investigate the potential of exact fixed-parameter algorithms for combinatorially hard graph problems under three criteria: design, analysis, and implementation. The main focus lay on the study of the power of various tools and techniques—such as data reduction, bounded search trees, graph separation, or tree decompositions—and their applicability to various (planar) graph problems. As running examples we concentrated on par-VERTEX COVER, par-INDEPENDENT SET, and par-DOMINATING SET, respectively, which—according to the intensity by which they appear in the literature—must be considered to be the three most fundamental parameterized graph problems representing the three most relevant parameterized complexity classes FPT, W[1], and W[2], respectively.

We conclude by giving a sketchy overview on the new results provided in this work, by hinting to ongoing work in this specific research area, and, finally, by describing open questions and horizons for future research.

7.1 Brief Summary of Results

This section is meant as a very brief one-page listing of the *main* new results presented in this work. For a more detailed summary of the results we refer to Section 1.4: We came up with ...

- ... data reduction rules for par-DOMINATING SET leading to a linear problem kernel for par-DOMINATING SET on planar graphs of size $335k$ (→ Chapter: "Data Reduction").
- ... a time $O(8^k n)$ search tree algorithm for par-DOMINATING SET on planar graphs based on degree-branching (→ Chapter: "Bounded Search Trees").
- ... the notion of glueable vertex selection problems that allows for a general method to obtain time $O(2^{O(\sqrt{k})} n)$ algorithms for parameterized problems on a graph class that admits a $\sqrt{\cdot}$-separator theorem; these include par-VERTEX COVER, par-INDEPENDENT SET, or par-DOMINATING SET on planar graphs. These fixed-parameter algorithms are asymptotically optimal, unless $3\,\text{SAT} \in \text{DTIME}(2^{o(n)})$, n being the number of variables (→ Chapter: "Graph Separation").

- ... a new geometric $\sqrt{\cdot}$-separator theorem for disk graphs of bounded radius ratio (→ Chapter: "Graph Separation").
- ... a time $2^{O(\sqrt{k}\log(n))}$ algorithm for par-INDEPENDENT SET on disk graphs of bounded radius ratio (→ Chapter: "Graph Separation").
- ... fixed-parameter algorithms with running time $O(2^{O(\sqrt{k})}+n)$ for par-INDEPENDENT SET, par-DOMINATING SET on disk graphs with ϑ-precision (→ Chapter: "Graph Separation").
- ... new constructive and asymptotically optimal upper bounds for the treewidth $\mathrm{tw}(G)$ of a planar graph G of the form $\mathrm{tw}(G) = O(\sqrt{\mathrm{vc}(G)})$ and $\mathrm{tw}(G) = O(\sqrt{\mathrm{ds}(G)})$ (where $\mathrm{vc}(G)$ and $\mathrm{ds}(G)$ denote the vertex cover number and the domination number, respectively) (→ Chapter: "Tree Decomposition Based Algorithms").
- ... the notion of "Layerwise Separation Property (LSP)" for parameterized planar graph problems that allows for efficient time $O(2^{O(\sqrt{k})}n)$ algorithms based on tree decompositions; the list of problems with LSP includes par-VERTEX COVER, par-INDEPENDENT SET, par-DOMINATING SET, par-INDEPENDENT DOMINATING SET, par-TOTAL DOMINATING SET, PERFECT DOMINATING SET, par-PERFECT CODE, or par-FACE COVER on planar graphs (→ Chapter: "Tree Decomposition Based Algorithms").
- ... improved dynamic programming on tree decompositions for domination-like problems; in particular, we exhibited a time $O(4^{\ell}N)$ algorithm for DOMINATING SET, INDEPENDENT DOMINATING SET, and PERFECT DOMINATING SET (where ℓ is the width of the underlying tree decomposition and N being the number of tree nodes), an $O(3^{\ell}N)$ algorithm for RED-BLUE DOMINATING SET, and an $O(5^{\ell}N)$ algorithm for TOTAL DOMINATING SET (→ Chapter: "Tree Decomposition Based Algorithms").
- ... the design and implementation of the software package called "FPT-toolbox"including data reduction procedures as well as tree decomposition based methods for various hard planar graph problems (→ Chapter: "Experimental Studies").
- ... a series of experimental studies and an empirical evaluation of the implemented algorithms (→ Chapter: "Experimental Studies").

7.2 Ongoing research

The results of this work were well received in the parameterized complexity community and already had some impact on ongoing work of other research groups. Especially our work [4, 8] on fixed-parameter algorithms exposing—seemingly for the first time—a running time behavior with a sublinear term in the exponent (i.e., that is of the form $2^{O(\sqrt{k})}n^{O(1)}$) received much attention and may be counted to one of the most often cited papers in parameterized complexity during the last year (see, e.g., [53, 70, 71, 87, 94, 96, 97, 98, 123, 129, 160, 182]). Regarding parameterized planar graph problems we, thus, initiated and opened new research horizons that might be grouped into three different categories.

Further improvements of current algorithms. Very recently, we have seen new contributions dealing with improvements on the running time of our time $O(2^{O(\sqrt{k})}n)$ algorithm for par-DOMINATING SET on planar graphs.[1] Recall that the constant hidden in the exponent we derived in our analysis was $c = 12\sqrt{17\log(3)} \approx 62$ (see Remark 5.4.9).

Kanj and Perkovic [123] recently reduced this constant to $c \approx 27$ by using a more fine-grained analysis of our methods. They slightly modified the concept of our partial layerwise separation (see Subsection 5.2.4.1). In their context only separators are used which separate so-called "non-shallow" components[2] of successive layers (instead of the layers themselves). With these modifications they show that every "yes"-instance of the problem admits a layerwise separation of width 3 and size $15k + \sqrt{k}$ (thus, improving our Corollary 5.2.17 which guaranteed width 3 and size $51k$). This yields an upper bound for the treewidth $\mathrm{tw}(G)$ of a planar graph G of the form

$$\mathrm{tw}(G) \leq 15.6\sqrt{\mathrm{ds}(G)} + 50,$$

$\mathrm{ds}(G)$ being the domination number of G (thus, improving our Corollary 5.2.26). Then, basically our Theorem 5.4.6 is used to derive a time $O(2^{27\sqrt{k}}n)$ algorithm for par-DOMINATING SET on planar graphs.

In a very recent paper due to Fomin and Thilikos [97], this running time was further improved to $O(2^{15.13\sqrt{k}}k + k^4 + n^3)$. Instead of giving a direct upper bound on the treewidth of a planar graph in terms of the domination number, Fomin and Thilikos use the concept of so-called branch decompositions (see [172] for details): The authors prove—making use of the heavy machinery of Robertson and Seymour's graph minor theory—a non-trivial and non-constructive upper bound of $\mathrm{bw}(G) \leq 3\sqrt{4.5\,\mathrm{ds}(G)}$ (here, $\mathrm{bw}(G)$ is the branchwidth of G), which is not far from the optimal (they provide a lower bound of $3\sqrt{\mathrm{ds}(G)} + O(1)$). In fact, as a consequence, since $\mathrm{tw}(G) \leq 1.5\,\mathrm{bw}(G)$ holds for every graph G with at least three edges [172], they even get the upper bound

$$\mathrm{tw}(G) \leq 4.5^{3/2}\sqrt{\mathrm{ds}(G)} \approx 9.55\sqrt{\mathrm{ds}(G)}.$$

The algorithmic consequences are as follows: In a first phase, Fomin and Thilikos apply our linear problem kernelization (see Theorem 2.2.1) which takes time $O(n^3)$. In a second phase, the fact is used that *on planar graphs* an optimal branch decomposition can be computed in time $O(n^4)$ [177]. Despite the high degree of the polynomial this algorithm is claimed to be practical [113, 114]. Moreover, in a third phase, according to the authors, a derivation of our new dynamic programming algorithm that was used to solve DOMINATING SET on a given tree decomposition (see Theorem 5.3.1) can be straight-forwardly adapted to solve the problem on a given branch decomposition in time $O(2^{(3\log_4 3)\ell}n)$ (where ℓ is an upper bound on the branchwidth of the given decomposition and n is the number of vertices in the graph). This results in a total running time of $O(2^{15.13\sqrt{k}}k + k^4 + n^3)$.

It is not very surprising that the constant derived in our analysis could be improved in a problem-specific manner for a problem like par-DOMINATING SET. The aim of our approach, in first place, was to give a *general* method that is applicable to various graph problems. For

[1] With the linear problem kernel as a preprocessing, we alternatively obtained a time $O(2^{O(\sqrt{k})}k+n^3)$ algorithm.

[2] A connected component of a layer is defined to be non-shallow if it has height at least $\lceil\sqrt{k}\rceil + 1$ in the layer decomposition tree (as defined in Subsection 5.1.2).

this purpose and for the sake of conceptual simplicity we deliberately sacrificed problem-specific fine-tuning of the analysis. Finally, we remark that the improved theoretical worst-case upper bounds of the exponential bases from [123] and [97] still are far away from being practical. Our (ultimate) goal should be the design of an algorithm with a worst-case upper bound on the running time of $O(2^{\sqrt{k}}n)$! The work initiated by us together with the presented improvements might be seen as a first step in this direction.

Extension of algorithm to broader graph classes. Some efforts have already been made to generalize some of our results on planar graphs to a wider class of graphs:

As a first example, we mention the recent work of Demaine *et al.* [71] who considered so-called α-recognizable clique-sum graphs (see [73]). An s-(clique-)sum is an operation for two graphs G_1 and G_2 that formally can be defined as follows: For $i = 1, 2$ find a clique $W_i \subseteq V(G_i)$ of size s. Choose a bijection $h : W_1 \to W_2$. Let G_i' be obtained from G_i by deleting some (possibly no) edges from $G_i[W_i]$. Then, the graph obtained from the disjoint union of G_1' and G_2' where each vertex $w \in W_1$ is identified with $h(w) \in W_2$ is called an i-(clique-)sum of G_1 and G_2. A graph class $\mathbb{G}$ is called a *clique-sum class* with defining pair $(\mathcal{C}, s)$ (here, $\mathcal{C}$ is a set of graphs and s is some integer), if each graph in $\mathbb{G}$ can be obtained by a finite set of s-(clique-)sum operations applied to planar graphs or graphs from $\mathcal{C}$. A clique-sum class is said to be α-recognizable if, for each graph, the corresponding set of clique-sum operations can be found in time $O(n^\alpha)$.

Demaine *et al.* carried our tree decomposition based approach (see Chapter 5) over to such clique-sum graph classes by using standard-techniques to compute a tree decomposition for clique-sums. In particular, they showed that the upper bound

$$\mathrm{tw}(G) \leq 6\sqrt{34\,\mathrm{ds}(G)} + \max\{8, d\}$$

(compare with our Corollary 5.2.26) also holds if G is from an α-recognizable clique-sum class $\mathbb{G}$ with defining pair $(\mathcal{C}, s)$. Here, $d = \max_{H \in \mathcal{C}} \mathrm{tw}(H)$ is called the *base* of $\mathbb{G}$. A corresponding tree decomposition can be constructed in time $O(n^\alpha)$. Using our Theorem 5.3.1, this results in a time $O(2^{12\sqrt{34k}}n + n^\alpha)$ algorithm for par-DOMINATING SET on an α-recognizable clique-sum class. Examples for such graphs are given by the class of graphs that exclude a single-crossing graph H as a minor which is a clique-sum class with base d_H that depends only on H [173]. In particular, the class of $K_{3,3}$-minor-free graphs is a clique-sum class with defining pair $(\{K_5\}, 2)$ and base $d = 4$ that is 1-recognizable [28]. Similarly, the class of K_5-minor-free graphs is a clique-sum class with defining pair $(\{V_8\}, 3)$[3] and base $d = 4$, that is 2-recognizable [126]. Since, by Kuratowski's theorem [159], planar graphs are equivalent to the set of graphs that exclude a $K_{3,3}$ *and* a K_5 as a minor, the results in [71] are a generalization of our results. Note, however, that a clique-sum graph with bounded base cannot contain arbitrary large cliques.

As a second example, we mention the work of Ellis *et al.* [87] who extended our search tree algorithm (see Theorem 3.2.3) to the class $\mathbb{G}(S_g)$ of graphs of bounded genus, i.e., of graphs that can be drawn without any edge-crossing on the surface S_g of genus g. The authors prove a branching theorem that is similar to the corresponding theorem for the planar case (see Theorem 3.2.10). The overall running time of their algorithm is $O(d(g)^k n^2)$, where $d(g) =$

[3]Here, V_8 is a cycle of length eight in which each pair of diagonally opposite vertices is joined by an edge.

$24g^2+24g+1$ (for $g \geq 1$) grows quadratically with the genus g. As in the planar case, their proof is based on topological arguments using the (generalized) Euler formula for graphs of bounded genus. Again, as for the graph classes studied in [71], we remark that the class $\mathbb{G}(S_g)$, for fixed g, cannot contain arbitrary large cliques. It would be interesting to see a fixed-parameter algorithm for par-DOMINATING SET on a non-trivial graph class which may contain arbitrary large cliques (as, e.g., the class of general disk graphs). Such a result so far is unknown.

Extension of algorithms to further problems. Since we provided first examples of fixed-parameter algorithms with a sublinear term in the exponent for several problems on planar graphs, the "pool" of such problems seems to increase steadily. Various papers are concerned with the design of fixed-parameter algorithms that have running time $2^{e(k)} \cdot n^{O(1)}$ where the function e grows sublinearly in the exponent. Mostly, the techniques used are a derivation of our tree decomposition based approach, where, in a first phase, one shows that a tree decomposition of width $O(e(k))$ (k being the problem parameter) can be constructed efficiently, and, in a second phase, one solves the problem in polynomial time on graphs of bounded treewidth. Such an approach is, e.g., followed in [129], for the par-FEEDBACK VERTEX SET or the par-DISJOINT CYCLE problems on planar graphs. In the first problem, one is given an undirected graph $G = (V, E)$ and a parameter k, and the task is to decide whether there is a vertex set $V' \subseteq V$ of size at most k such that each cycle of G contains at least one point in V'. In the latter problem, one is given an undirected graph $G = (V, E)$ and a parameter k and the question is whether there are at least k vertex disjoint cycles in G. For both problems, an algorithm with running time $O(2^{O(\sqrt{k}\log(k))} n)$ is given [129].

These were improved by Demaine *et al.* [71] who gave time $O(2^{O(\sqrt{k})} n + n^{\alpha})$ algorithms for these problems on α-recognizable clique-sum classes. Moreover, Demaine *et al.* address various other graph problems on clique-sum classes which all allow for such "sublinear-exponential" fixed-parameter algorithms. The list of problem studied there includes par-Y-DOMINATING SET, par-EDGE DOMINATING SET, par-CLIQUE-TRANSVERSAL SET, par-MINIMUM MAXIMAL MATCHING, par-KERNEL on digraphs, and various so-called vertex removal problems (see [71] for details). Most notably, all results are established by our fundamental upper bound $\mathrm{tw}(G) = O(\sqrt{\mathrm{ds}(G)})$.

7.3 Open Problems and Future Research

This work provides various starting points for future research. We formulate some of the prospects in this area.

It is a challenging project to continue the ongoing work (see Section 7.2) on the design of "sublinear-exponential" fixed-parameter algorithms, i.e., fixed-parameter algorithms which have a sublinear term in the exponent: In first place, ameliorating some of the admittedly bad worst-case constants in the time $O(2^{O(\sqrt{k})} n)$ algorithms needs and deserves future work. In particular, it would be interesting to know whether the more elusive par-DOMINATING SET problem on planar graphs admits an exponential base comparable to the one derived for the seemingly much less elusive par-VERTEX COVER problem on planar graphs. The two "follow-up" papers [71, 123] can only be considered as a first step in this direction. Secondly, just as we did for planar graphs, we should find other graph classes which allow for general methods to design sublinear-exponential fixed-parameter algorithms. Maybe our concept of the Layerwise Separation Property (LSP) can be carried over to, e.g., graphs of bounded genus. A first attempt to extend the LSP to a wider class of graphs was provided by Demaine *et al.* [71] who considered graphs of locally bounded treewidth. Possibly, an approach based on other decomposition techniques, such as branch decompositions [97], is even more fertile in this respect. Thirdly, the list of problems that allow for sublinear-exponential fixed-parameter algorithms should be further completed. At the same time and to a similar extent the studies on lower bounds (see Subsection 4.2.2) for such problems is an area that is worth further consideration.

Our investigations on sublinear-exponential fixed-parameter algorithms, together with the lower bounds given in Subsection 4.2.2 stirred up new questions concerning the structure of the complexity class FPT: The notion of fixed-parameter tractability is maybe—in a sense—inadequate from a practical point of view since it allows *arbitrary* exponential growth in the exponential running time component for the parameter k. It is clear that for a fixed-parameter algorithm with running time $f(k) \cdot n^{O(1)}$, a function $f(k) = 2^{O(k)}$ is more desirable than, e.g., a function $f(k) = 2^{O(2^{2^{2^k}})}$. However, this is not taken care of in the notion of fixed-parameter tractability as it is. Besides, there is a new structural aspect within the class FPT: We can solve par-DOMINATING SET on planar graphs in time $O(2^{O(\sqrt{k})}n)$ whereas, according to Theorem 4.2.10, par-VERTEX COVER on general graphs does not allow for such an algorithm unless the unlikely fact that 3 SAT $\in$ DTIME$(2^{o(n)})$ holds, where n is the number of variables of a 3 SAT formula. These results might motivate the following refinement of the class FPT according to the asymptotic behavior of the exponential term in the running time:

$$\mathrm{FPT}(g) := \left\{ \mathcal{L} \subseteq \Sigma^* \times \mathbb{N} \;\middle|\; \begin{array}{l} \exists \text{ an algorithm to decide } (x,k) \in \mathcal{L} \text{ in time} \\ 2^{O(g)} \cdot n^{\alpha}, \text{ for some } \alpha \in \mathbb{R}^+ \end{array} \right\}.$$

It might be interesting to exploit a hierarchy in these classes and to investigate a completeness program for these classes FPT(g)—similar to the studies of Impagliazzo, Pazuri, and Zane [119] on strongly exponential problems. The corresponding notion of reduction would be the refined parameterized reduction (see Definition 4.2.8) which precisely takes care of the way the parameter k is transformed.

Our experiments with combinatorially generated random graphs revealed the great potential of data reduction by problem kernelization. Specifically, data reduction based on simple (and easy to implement) reduction rules—such as the ones for par-DOMINATING SET (see Subsection 2.2.1)—seemed to be very successful and competitive. Clearly, the worst-case bound on the kernel size for par-DOMINATING SET on planar graphs needs further improvement, too. This could be done by providing further (more fine-grained) reduction rules that take into account not only the local structure given by two vertices, but also by considering the neighborhood of some fixed number of vertices. We deliberately neglected these generalizations, for the proof of the linear problem kernel would have become even more intricate and lengthy. Besides, since we used only topological arguments, a straight-forward generalization of a linear problem kernel for graphs of genus bounded by some constant g seems possible. Then, a similar argument as in Remark 4.2.3, would establish an $O(c^{\sqrt{gk}}n)$ algorithm which would improve on [87]. In addition, it could be a promising project to design—on a purely heuristic base or even with provable worst-case upper bounds—similar reduction rules, e.g., for par-VERTEX COVER. The hope is that such rules, from a practical point of view, could even outperform the data reduction due to Nemhauser and Trotter. Moreover, it is still a long-standing open problem whether par-VERTEX COVER on *planar* graphs has a problem kernel of size smaller than $2k$. Maybe the planarity assumption can be exploited in Nemhauser and Trotter's theorem? Moreover, it seems likely that the Nemhauser and Trotter theorem could be generalized to hypergraphs to improve the problem kernel for the parameterized d-HITTING SET problem (the generalization of VERTEX COVER to hypergraphs) [158]. Finally, are there lower bound results on problem kernels? Can we find a non-artificial parameterized problem for which a problem kernel of size $o(p(n))$, where $p(n)$ is some polynomial, cannot exist unless some unlikely complexity-theoretical collapses hold true? Efficient data reduction for parameterized problems probably has the potential to develop into a research area on its own.

Most of our fixed-parameter results concerning par-INDEPENDENT SET on planar graphs make use of the linear problem kernel. This problem kernel was obtained by using the four-color theorem which already guarantees that we can find an independent set of size $\lceil \frac{n}{4} \rceil$. In a way—even though perfectly correct from a theoretical point of view—this is not in the sense of designing parameterized algorithms, where we assume the problem parameter k (here, the size of an optimal independent set) to be small. A more reasonable and maybe more honest formulation of par-INDEPENDENT SET on planar graphs would be given by parameterizing above this guaranteed value (see Remark 2.1.6). Given a parameter k, the question then becomes whether there exists an independent set of size at least $\lceil \frac{n}{4} \rceil + k$. With respect to this parameterization, the parameterized complexity of INDEPENDENT SET on planar graphs still is open.

Dealing with disk graphs (see Section 4.3), we opened a new source for parameterized study. While we now well understand the parameterized complexity of our threesome of graph problems (par-VERTEX COVER, par-INDEPENDENT SET, and par-DOMINATING SET) on ϑ-precision disk graphs, the situation is less clear for the class $\mathbb{DG}_\rho$ of disk graphs bounded radius ratio, or even for general disk graphs $\mathbb{DG}$. We came up with a time $2^{O(\sqrt{k}\log(n))}$ algorithm for par-INDEPENDENT SET on $\mathbb{DG}_\rho$, yet, the parameterized complexity (i.e., whether the problem is in FPT or W[1]-complete) is still open. We want to emphasize that we are not aware of a (non-artificial) W[1]-hard problem which can be solved in time bounded by an exponential with a sublinear exponent. Similarly, its W[1]-hardness would expose—to our knowledge—the first example of a fixed-parameter intractable problem that simultaneously allows a PTAS (where the problem parameter and the the approximation guarantee refer to the same cost function). Finally, from an application point of view, it might be interesting to study parameterized graph problems on other geometric graphs such as, e.g., intersection graphs of rectangles (the par-INDEPENDENT SET problem on these graphs is important for applications like map labeling [2]). It seems possible that our techniques based on geometric separation carry over to these graphs as well.

In order to obtain statistically relevant empirical results more extensive experimental studies on a broader range of test samples (including more real world data) would be necessary. Tuning algorithms for real applications, further aspects should be taken into consideration. E.g., so far we hardly dealt with the memory requirements for our algorithms, yet this might be a crucial bottleneck in practice. In particular, space consumption for tree decomposition based algorithms (see [29]) is a so far widely underrated area to which hardly any innovative contributions were made. Generally speaking, in designing fixed-parameter algorithms as a theorist we always tend to focus on provable worst-case upper bounds. In order to derive an optimal performance behavior *in practice* more knowledge on the actual input data (arising from a given application) should be collected and further (heuristic) fine-tuning should be developed and incorporated.

Bibliography

[1] K. A. Abrahamson, R. G. Downey, and M. R. Fellows. Fixed-parameter tractability and completeness IV: On completeness for W[P] and PSPACE analogs. *Annals of Pure and Applied Logic*, **73:**235–276, 1995. 3, 76

[2] P. K. Agarwal, M. van Kreveld, and S. Suri. Label placement by maximum independent set in rectangles. *Computational Geometry: Theory and Applications*, **11(3–4):**209–218, 1998. 10, 78, 159

[3] J. Alber. On implemented semigroups. *Semigroup Forum*, **63(3)**:371–386, 2001. 173

[4] J. Alber, H. L. Bodlaender, H. Fernau, T. Kloks, and R. Niedermeier. Fixed parameter algorithms for dominating set and related problems on planar graphs. *Algorithmica*, **33(4):**461–493, 2002.
Extended abstract (of the authors J. Alber, H. L. Bodlaender, H. Fernau, and R. Niedermeier) in *Proceedings 7th SWAT*, Springer-Verlag LNCS 1851, pages 97–110, 2000. 99, 100, 102, 104, 105, 113, 123, 130, 154, 162, 173

[5] J. Alber, F. Dorn, and R. Niedermeier. Experimental evaluation of a tree decomposition based algorithm for vertex cover on planar graphs. Manuscript, March 2002. Accepted for publication in *Discrete Applied Mathematics*. 137, 139, 173

[6] J. Alber, H. Fan, M. R. Fellows, H. Fernau, R. Niedermeier, F. Rosamond, and U. Stege. Refined search tree technique for dominating set on planar graphs. In *Proceedings 26th MFCS*, Springer-Verlag LNCS 2136, pages 111–122, 2001.
Long version submitted to *Journal of Computer and System Sciences.* 46, 49, 173

[7] J. Alber, M. R. Fellows, and R. Niedermeier. Efficient data reduction for dominating set: a linear problem kernel for the planar case. In *Proceedings 8th SWAT*, Springer-Verlag LNCS 2368, pages 150–159, 2002. 22, 173

[8] J. Alber, H. Fernau, and R. Niedermeier. Parameterized complexity: exponential speed-up for planar graph problems. In *Proceedings 28th ICALP*, Springer-Verlag LNCS 2076, pages 261–272, 2001.
Long version available as Technical Report TR01-023, Electronic Colloquium on Computational Complexity (ECCC), Trier, March 2001. 99, 102, 154, 173

[9] J. Alber, H. Fernau, and R. Niedermeier. Graph separators: a parameterized view. In *Proceedings 7th COCOON*, Springer-Verlag LNCS 2108, pages 318–327, 2001.
Long version available as Technical Report WSI-2001-8, Wilhelm-Schickard-Institut für Informatik, Universität Tübingen. Accepted for publication in *Journal of Computer and System Sciences.* 67, 68, 73, 75, 173

[10] J. Alber and J. Fiala. Geometric separation and exact solutions for independent set on disk graphs. In *Proceedings 2nd IFIP TCS (Foundations of Information Technology in the Era of Network and Mobile Computing)*, Kluwer Academic Publishers, pages 26–37, 2002.

Long version available as Technical Report ITI Series 2002-79, Institute for Theoretical Computer Science, Charles University, Prague. 79, 173

[11] J. Alber, J. Gramm, J. Guo, and R. Niedermeier. Towards optimally solving the longest common subsequence problem with nested arc annotations in linear time. In *Proceedings 13th CPM*, Springer-Verlag LNCS 2373, pages 99–114, 2002. 173

[12] J. Alber, J. Gramm, and R. Niedermeier. Faster exact solutions for hard problems: a parameterized point of view. *Discrete Mathematics*, **229(1)**: 3–27, 2001. 3, 173

[13] J. Alber and R. Niedermeier. On multi-dimensional curves with Hilbert property. *Theory of Computing Systems*, **33(4)**:295–312, 2000.
Extended abstract under the title "On multi-dimensional Hilbert indexings" in *Proceedings 4th COCOON*, Springer-Verlag LNCS 1449, pages 329–338, 1998. 173

[14] J. Alber and R. Niedermeier. Improved tree decomposition based algorithms for domination-like problems. In *Proceedings 5th LATIN*, Springer-Verlag LNCS 2286, pages 613–627, 2002. 112, 114, 173

[15] J. Alber and R. Niedermeier. *Parametrisierte Algorithmen*. Lecture notes and slides (in German). WSI für Informatik, Universität Tübingen, February 2001. 3, 112, 173

[16] J. Alber and R. Niedermeier. Improved tree decomposition based algorithms for parameterized planar dominating set. Reading guide for [4]. In KAM-DIMATIA Series, report no. 2001-514, Faculty of Mathematics and Physics, Charles University, Praha, Czech Republic, 2001. 99, 173

[17] L. G. Aleksandrov and H. N. Djidjev. Linear algorithms for partitioning embedded graphs of bounded genus. *SIAM Journal on Discrete Mathematics*, **9**:129–150, 1996. 65

[18] N. Alon, P. D. Seymour, and R. Thomas. A separator theorem for graphs with an excluded minor and its applications. In *Proceedings 22nd ACM STOC*, pages 293–299, ACM Press, 1990. 65

[19] N. Alon, P. D. Seymour, and R. Thomas. A separator theorem for nonplanar graphs. *Journal of the AMS*, **3**:801–808, 1990. 65

[20] N. Alon, P. D. Seymour, and R. Thomas. Planar separators. *SIAM Journal on Discrete Mathematics*, **2**:184–193, 1990. 65, 107

[21] K. Appel, W. Haken. Every planar map is four-colorable. Part I. Discharging. *Illinois J. Math.*, **21**:429–490, 1977. 20

[22] K. Appel, W. Haken. Every planar map is four-colorable. Part II. Reducibility. *Illinois J. Math.*, **21**:491–567, 1977. 20

[23] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, **8**:277–284, 1987. 97

[24] S. Arnborg, D. G. Corneil, A. Proskurowski, and D. Seese. An algebraic theory of graph reduction. *Journal of the ACM*, **40**:1134–1164, 1993. 112

[25] S. Arnborg and A. Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Discrete Applied Mathematics*, **23(1)**:11–24, 1989. 112

[26] S. Arnborg, A. Proskurowski, and D. G. Corneil. Forbidden minors characterization of partial 3-trees. *Discrete Mathematics*, **80**:1–19, 1990. 96

[27] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, **45**:501–555, 1998. 11

[28] T. Asano. An approach to the subgraph homeomorphism problem. *Theoretical Computer Science*, **38(2-3)**:249–267, 1985. 156

[29] B. Aspvall, A. Proskurowski, and J. A. Telle. Memory requirements for table computations in partial k-tree algorithms. *Algorithmica*, **27**: 382–394, 2000. 137, 142, 159

[30] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation.* Springer-Verlag, 1999. 1, 11

[31] B. S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM*, **41(1)**:153–180, 1994. 9, 10, 12, 22, 89, 97, 124

[32] R. Balasubramanian, M. R. Fellows, and V. Raman. An improved fixed parameter algorithm for vertex cover. *Information Processing Letters*, **65(3)**:163–168, 1998. 9, 43

[33] R. Bar-Yehuda and S. Even. A local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Discrete Mathematics*, **25**:27–46, 1985. 9, 20, 22

[34] M. Bellare, O. Goldreich, and M. Sudan. Free bits, PCPs and non-approximability towards tight results. *SIAM Journal on Computing*, **27**:804–915, 1998. 10

[35] C. Berge. *Graphs and Hypergraphs.* American Elsevier, 1973. 11

[36] M. W. Bern, E. L. Lawler, and A. L. Wong. Linear-time computation of optimal subgraphs of decomposable graphs. *Journal of Algorithms*, **8**:216–235, 1987. 137, 142

[37] M. Bern and A. Sahai. Pushing disks together—the continuous-motion case. *Discrete and Computational Geometry*, **20(4)**:499–514, 1998. 85

[38] D. A. Berque, J. A. Edmonds, and M. K. Goldberg. Implementing progress indicators for recursive algorithms. In *Proceedings 8th ACM/SIGAPP Symposium on Applied Computing (SAC)*, pages 533–538, 1993. 137

[39] D. Bienstock and C. L. Monma. On the complexity of covering vertices by faces in a planar graph. *SIAM Journal on Computing*, **17**:53–76, 1988. 13, 120

[40] D. Bienstock and C. L. Monma. On the complexity of embedding planar graphs to minimize certain distance measures. *Algorithmica*, **5**:93–109, 1990. 98

[41] H. L. Bodlaender. Dynamic programming on graphs of bounded treewidth. In *Proceedings 15th ICALP*, Springer-Verlag LNCS 317, pages 105–118, Springer-Verlag, 1988. 112

[42] H. L. Bodlaender. A tourist guide through treewidth. In *Acta Cybernetica*, **11**:1–23, 1993. 101

[43] H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, **25**:1305–1317, 1996. 97

[44] H. L. Bodlaender. Treewidth: Algorithmic techniques and results. In *Proceedings 22nd MFCS*, Springer-Verlag LNCS 1295, pages 19–36, 1997. 112

[45] H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, **209**:1–45, 1998. 65, 94, 100, 101, 102

[46] H. L. Bodlaender, J. R. Gilbert, H. Hafsteinsson, and T. Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *Journal of Algorithms*, **18(2)**:238–255, 1995. 97

[47] H. L. Bodlaender and F. V. Fomin. Tree decompositions with small cost. In *Proceedings 8th SWAT*, Springer-Verlag LNCS 2368, pages 378–387, 2002. 142

[48] R. Boppana and M. M. Halldórsson. Approximating maximum independent sets by excluding subgraphs. *BIT*, **32(2)**:180–196, 1992. 10

[49] V. Bouchitté, D. Kratsch, H. Müller, and I. Todinca. On treewidth approximations. *Cologne-Twente Workshop on Graphs and Combinatorial Optimization (CTW)*, 2001. 97, 137

[50] A. Brandstädt, V. B. Le, and J. P. Spinrad. *Graph Classes: a Survey.* SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, 1999. 4

[51] J. F. Buss and J. Goldsmith. Nondeterminism within P. *SIAM Journal on Computing*, **24**:873–921, 1993. 21, 22

[52] M. Cadoli, F. M. Donini, P. Liberatore, and M. Schaerf. Preprocessing of intractable problems. *Information and Computation*, **176**:89–120, 2002. 17

[53] L. Cai and D. Juedes. On the existence of subexponential parameterized algorithms. Manuscript, submitted for publication. October 2001. Revised version of: Subexponential parameterized algorithms collapse the W-hierarchy. In *Proceedings 28th ICALP*, Springer-Verlag LNCS 2076, pages 273–284, 2001. This conference version contains some major flaws, see [84]. 3, 76, 77, 154

[54] T. Y. Chang and W. E. Clark. The domination number of the $5 \times n$ and the $6 \times n$ grid graphs. *Journal of Graph Theory*, **17**:81–107, 1993. 12

[55] T. Y. Chang, W. E. Clark, and E. O. Hare. Domination numbers of complete grid graphs I. *Ars Combinatorica*, **38**:97–111, 1994. 12

[56] J. Chen, I. A. Kanj, and W. Jia. Vertex cover: further observations and further improvements. *Journal of Algorithms*, **41**:280–301, 2001. 9, 20, 21, 43

[57] Z.-Z. Chen. Approximation algorithms for independent sets in map graphs. *Journal of Algorithms*, **41**:20–40, 2001. 65

[58] Z.-Z. Chen, M. Grigni, and C. Papadimitriou. Map graphs. *Journal of the ACM*, **49(2)**:127–138, 2002. 65

[59] N. Chiba, T. Nishizeki, S. Abe, and T. Ozawa. A linear algorithm for embedding planar graphs using PQ-trees. *Journal of Computer and System Sciences*, **30**:54–76, 1985. 5

[60] N. Chiba, T. Nishizeki, and N. Saito. A linear 5-coloring algorithm of planar graphs. *Journal of Algorithms*, **2**:317–327, 1981. 20

[61] B. Chor and M. Sudan. A geometric approach to betweenness. *SIAM Journal on Discrete Mathematics*, **11(4)**:511–523, 1998. 21

[62] B. N. Clark, C. J. Colbourn, and D. S. Johnson. Unit disk graphs. *Discrete Mathematics*, **86**:165–177, 1990. 12, 89

[63] E. J. Cockayne, E. O. Hare, S. T. Hedetniemi, T. V. Wimer. Bounds for the domination number of grid graphs. *Congressus Numerantium*, **47**:217–228, 1985. 12, 112

[64] S. A. Cook and R. A. Reckhow. Time bounded random access machines. *Journal of Computer and System Sciences*, **7**:354–375, 1976. 6

[65] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* The MIT Press, 2nd edition, 2001. 6

[66] D. G. Corneil and J. M. Keil. A dynamic programming approach to the dominating set problem on k-trees. *SIAM Journal on Algebraic Discrete Methods*, **8**: 535–543, 1987. 112, 113

[67] B. Courcelle. Graph rewriting: an algebraic and logic approach. In *Handbook of Theoretical Computer Science, Vol. B: Formal Models and Semantics*, pages 193–242. North Holland, 1990. 96, 112

[68] P. Crescenzi and V. Kann. How to find the best approximation results—a follow-up to Garey and Johnson. *ACM SIGACT News*, **29(4)**:90–97, 1998. 9, 11

[69] F. Dehne, A. Rau-Chaplin, U. Stege, and P. J. Taillon. Solving large FPT problems on coarse grained parallel machines. Manuscript, July 2001. Submitted to *Journal of Computer and System Sciences.* 142

[70] E. D. Demaine, M. T. Hajiaghayi, and D. M. Thilikos. 1.5-Approximation for treewidth of graphs excluding a graph with one crossing as a minor. In *Proceedings 5th APPROX 2002*, Springer-Verlag LNCS 2462, pages 67–80, 2002. 97, 154

[71] E. D. Demaine, M. T. Hajiaghayi, and D. M. Thilikos. Exponential speedup of fixed-parameter algorithms on $\mathsf{K}_{3,3}$-minor-free or K_5-minor-free graphs To appear in *Proceedings 13th ISAAC*, Springer-Verlag LNCS, Vancouver, Canada, December 2002. Preliminary version appear as Technical report LSI-02-29-R , Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, Barcelona, Spain, 2002. 154, 156, 157, 158

[72] G. Di Battista, P. Eades, R. Tamassia, and I.G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice Hall, 1999. 126

[73] R. Diestel. Simplicial decompositions of graphs: a survey of applications. *Discrete Mathematics*, **75**:121–144, 1989. 156

[74] R. Diestel. *Graph Theory.* Springer-Verlag, 2nd edition, 2000. 4

[75] H. N. Djidjev. On the problem of partitioning planar graphs. *SIAM J. Algebraic Discrete Methods*, **3(2)**:229–240, 1982. 65

[76] H. N. Djidjev. A separator theorem for graphs of fixed genus. *Serdica*, **11**:319–329, 1985. 65, 75

[77] H. N. Djidjev and S. Venkatesan. Reduced constants for simple cycle graph separation. *Acta Informatica*, **34**:231–243, 1997. 65, 74

[78] F. Dorn. *Tuning Algorithms for Hard Graph Problems.* Study work in preparation, Universität Tübingen, Germany. 2002. 136, 137

[79] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness. *Congressus Numerantium*, **87**:161–187, 1992. 12

[80] R. G. Downey and M. R. Fellows. Parameterized computational feasibility. In *P. Clote, J. Remmel (eds.): Feasible Mathematics II*, pages 219–244. Birkhäuser, 1995. 12, 46

[81] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness II: On completeness for W[1]. *Theoretical Computer Science*, **141**:109–131, 1995. 10, 91, 92

[82] R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Monographs in Computer Science, Springer-Verlag, 1999. II, 2, 3, 4, 7, 9, 10, 12, 13, 14, 46, 91, 92

[83] R. G. Downey and M. R. Fellows. Parameterized complexity after (almost) ten years: Review and open questions. In *Combinatorics, Computation & Logic (DMTCS and CATS'99)*, Australian Computer Science Communications, **21(3):**1–33, Springer-Verlag Singapore, 1999. 3

[84] R. G. Downey, M. R. Fellows, R. Niedermeier, and P. Rossmanith (eds.). *Parameterized Complexity.* Dagstuhl-Seminar-Report No. 316, August 2001. 3, 164

[85] R. G. Downey, M. R. Fellows, and U. Stege. Parameterized complexity: A framework for systematically confronting computational intractability. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, **49**:49–99, 1999. 3, 9

[86] F. v. d. Eijkhof and H. L. Bodlaender. Safe reduction rules for weighted treewidth. To appear in *Proceedings 28th WG*, Springer-Verlag LNCS, Cesky Krumlov, Czech Republic, June 2002. 97

[87] J. Ellis, H. Fan, and M. R. Fellows. The dominating set problem is fixed parameter tractable for graphs of bounded genus. In *Proceedings 8th SWAT*, Springer-Verlag LNCS 2368, pages 180-189, 2002. 154, 156, 158

[88] D. Eppstein, G. L. Miller, and S. H. Teng. A deterministic linear time algorithm for geometric separators and its applications. In *Proceedings 9th ACM Symposium on Computational Geometry*, pages 99–108, 1993. 78, 84

[89] T. Erlebach, K. Jansen, and E. Seidel. Polynomial-time approximation schemes for geometric graphs. In *Proceedings 12th ACM-SIAM SODA*, pages 671–679, 2001. 89

[90] I. Fáry. On straight line representation of planar graphs. *Acta Scientiarum Mathematicarum*, **11**:229–233, 1948. 79

[91] U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, **45**:634–652, 1998. 11

[92] M. R. Fellows. Parameterized complexity: the main ideas and some research frontiers. In *Proceedings 12th ISAAC*, Springer-Verlag LNCS 2223, pages 291–307, 2001. 3

[93] M. R. Fellows and M. A. Langston. Nonconstructive advances in polynomial-time complexity. *Information Processing Letters*, **28**:157–162, 1987. 9

[94] H. Fernau. graph separator algorithms: a refined analysis. To appear in *Proceedings 28th WG*, Springer-Verlag LNCS, Cesky Krumlov, Czech Republic, June 2002. 75, 154

[95] H. Fernau and R. Niedermeier. An efficient exact algorithm for constraint bipartite vertex cover. *Journal of Algorithms*, **38(2)**:374–410, 2001. 9

[96] F. V. Fomin and D. M. Thilikos. New upper bounds on the decomposability of planar graphs and fixed parameter algorithms. Technical report LSI-02-56-R, Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, Barcelona, Spain, 2002. 154

[97] F. V. Fomin and D. M. Thilikos. Dominating sets in planar graphs: branch-width and exponential speed-up. To appear in *Proceedings 14th ACM-SIAM SODA*, Baltimore, MD, USA, 2003. 106, 154, 155, 156, 158

[98] M. Frick, and M. Grohe. Deciding first-order properties of locally tree-decomposable structures. *Journal of the ACM*, **48**:1184–1206, 2001. 12, 154

[99] M. Galota, C. Glaßer, S. Reith, and H. Vollmer. A polynomial-time-approximation scheme for base station positioning in UMTS networks. In *Proceedings 5th Discrete Algorithms and Methods for Mobile Computing and Communications*, pages 52–59, 2001. 11

[100] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness.* Freeman, 1979. 9, 10, 11

[101] M. Garey, D. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, **1**:237–267, 1976. 76, 77

[102] C. Glaßer, S. Reith, and H. Vollmer. The complexity of base station positioning in cellular networks. In *Proceedings of the ICALP Workshops*, Carleton Press, pages 167–177, 2000. 11

[103] D. H. Greene and D. E. Knuth. *Mathematics for the Analysis of Algorithms.* Progress in Computer Science and Applied Logic, Birkhäuser, 3rd edition, 1990. 41

[104] M. Grohe. Local tree-width, excluded minors, and approximation algorithms. Manuscript, January 2000. To appear in *Combinatorica*. 9, 10, 12

[105] T. Hagerup. Personal communication. July, 2002. 49

[106] F. Harary, R. Z. Norman, and D. Cartwright. *Structural Models: An Introduction to the Theory of Directed Graphs.* John Wiley, 1965. 11

[107] E. O. Hare, W. R. Hare, and S. T. Hedetniemi. Algorithms for computing the domination number of the $k \times n$ complete grid graph. *Congressus Numerantium*, **55**:81–92, 1986. 12

[108] J. Håstad. Some optimal inapproximability results. *Journal of the ACM*, **48**:798–859, 2001. 9

[109] T. W. Haynes. Domination in graphs: a brief overview. *Journal of Combinatorial Mathematics and Combinatorial Computing*, **24**:225–237, 1997. 11

[110] T. W. Haynes, S. T. Hedetniemi, and P. J. Slater. *Fundamentals of Domination in Graphs*. Monographs and textbooks in Pure and Applied Mathematics Vol. 208, Marcel Dekker, 1998. 7, 11, 12

[111] T. W. Haynes, S. T. Hedetniemi, and P. J. Slater (eds.). *Domination in Graphs; Advanced Topics*. Monographs and textbooks in Pure and Applied Mathematics Vol. 209, Marcel Dekker, 1998. 11, 168

[112] M. A. Henning. Domination in graphs, a survey. *Surveys in graph theory. Congressus Numerantium*, **116**:139–179, 1996. 11

[113] I. V. Hicks. *Branch Decompositions and their Applications.* PhD thesis, Rice University, 2000. 155

[114] I. V. Hicks. Planar branch decompositions. Manuscript, 2002. Submitted to *Discrete Applied Mathematics*. 155

[115] E. Dantsin, A. Goerdt, E. A. Hirsch, R. Kannan, J. Kleinberg, C. Papadimitriou, P. Raghavan, and U. Schöning. A deterministic $(2 - 2/(k+1))^n$ algorithm for k-SAT based on local search. To appear in *Theoretical Computer Science*, 2001. 3, 91

[116] D. S. Hochbaum, editor. *Approximation algorithms for NP-hard problems.* Boston, MA: PWS Publishing Company, 1997. 1, 11, 22

[117] J. Hopcroft and R. E. Tarjan. Efficient planarity testing. *Journal of the ACM*, **21(4)**:549–568, 1974. 5

[118] H. B. Hunt, M. V. Marathe, V. Radhakrishnan, S. S. Ravi, D. J. Rosenkrantz, and R. E. Stearns. NC-approximation schemes for NP– and PSPACE–hard problems for geometric graphs. *Journal of Algorithms*, **26**:238–274, 1998. 78, 80, 89

[119] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, **63(4)**:512–530, 2001. 63, 76, 79, 91, 92, 158

[120] M. S. Jacobson and L. F. Kinch. On the domination number of products of graphs: I. *Ars Combinatorica*, **18**:33–44, 1984. 12

[121] C. Jin, Q. Chen, and S. Jamin. Inet: internet topology generator. Technical Report Research Report CSE-TR-433-00, University of Michigan at Ann Arbor, 2000. 138

[122] A. Kanevsky. Finding all minimum-size separating vertex sets in a graph. *Networks*, **23**:533–541, 1993. 110

[123] I. A. Kanj and L. Perkovic. Improved parameterized algorithms for planar dominating set. In *Proceedings 27th MFCS 2002*, Springer-Verlag LNCS 2420, pages 399–410, 2002. 12, 106, 154, 155, 156, 157

[124] R. M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, Plenum Press, pages 85–103, 1972. 9

[125] L. L. Kelleher. *Domination in Graphs and its Application to Social Network Theory.* PhD thesis, Northeastern University, Boston, 1985. 11

[126] A. Kézdy and P. McGuiness. Sequential and parallel algorithms to find a K_5 minor. In *Proceedings 3rd ACM-SIAM SODA*, pages 345–356, 1992. 156

[127] S. Khuller. Algorithms column: the vertex cover problem. *ACM SIGACT News*, **33(2)**:31–33, 2002. 21

[128] T. Kloks. *Treewidth: Computations and Approximations*. Springer-Verlag LNCS 842, 1994. 94, 95

[129] T. Kloks, C. M. Lee, and J. Liu. New algorithms for k-face cover, k-feedback vertex set, and k-disjoint cycles on plane and planar graphs. To appear in *Proceedings 28th WG*, Springer-Verlag LNCS, Cesky Krumlov, Czech Republic, June 2002. 154, 157

[130] P. Koebe. Kontaktprobleme der konformen Abbildung. *Ber. Verh. Saechs. Akad. Leipzig*, **88**:141–164, 1936. 78, 79

[131] A. M. C. A. Koster, H. L. Bodlaender, and S. P. M. Hoesel. Treewidth: computational experiments. *Electronic Notes in Discrete Mathematics 8*, Elsevier Science Publishers, 2001. 97, 137

[132] D. Kratsch. Algorithms. Chapter 8 in [111]. 12, 13

[133] S.-Y. Kuo and W.K. Fuchs. Efficient spare allocation for reconfigurable arrays. *IEEE Design and Test*, **4**:24–31, 1987. 9

[134] J. van Leeuwen. Graph algorithms. In *Handbook of Theoretical Computer Science, Vol. A: Algorithms and Complexity*, pages 525–631. North Holland, 1990. 100

[135] S. Lehnert. *Experimental Analysis of a Search Tree Algorithm for Dominating Set.* Study work in preparation, Universität Tübingen, Germany. 2002. 143

[136] N. Lev-Tov and D. Peleg. Exact algorithms and approximation schemes for base station placement problems. In *Proceedings 8th SWAT*, Springer-Verlag LNCS 2368, pages 90–99, 2002. 92

[137] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM Journal of Applied Mathematics*, **36(2)**:177–189, 1979. 65, 92

[138] R. J. Lipton and R. E. Tarjan. Applications of a planar separator theorem. *SIAM Journal on Computing*, **9(3)**:615–627, 1980. 66, 91

[139] M. Livingston and Q. Stout. Constant time computation of minimum dominating sets. *Congressus Numerantium*, **105**:116–128, 1994. 12

[140] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM*, **41**:960–981, 1994. 12

[141] M. Mahajan and V. Raman. Parameterizing above guaranteed values: MaxSat and MaxCut. *Journal of Algorithms*, **31**:335–354, 1999. 21

[142] E. Malesińska. *Graph-Theoretical Models for Frequency Assignment Problems.* PhD thesis, Technische Univerität Berlin, 1997. 10, 78

[143] M. V. Marathe, H. Breu, H. B. Hunt III, S. S. Ravi, and D. J. Rosenkrantz. Simple heuristics for unit disk graphs. *Networks*, **25**:59–68, 1995. 89

[144] T. Matsui. Approximation algorithms for maximum independent set problems and fractional coloring problems on unit disk graphs. In *Proceedings 2nd JCDCG*, Springer-Verlag LNCS 1763, pages 194–200, 2000. 89

[145] K. Mehlhorn and S. Näher. *LEDA: A Platform of Combinatorial and Geometric Computing*. Cambridge University Press, Cambridge, England, 1999. 133, 136, 138

[146] Z. Michalewicz and D. B. Fogel. *How to solve it: Modern Heuristics*. Springer-Verlag, 2000. 1

[147] G. L. Miller, S. H. Teng, and S. A. Vavasis. A unified geometric approach to graph separators. In *Proceedings 32nd IEEE FOCS*, pages 538–547, 1991. 78, 84

[148] G. L. Miller, S. H. Teng, W. Thurston, and S. A. Vavasis. Separators for sphere packings and nearest neighbor graphs. *Journal of the ACM*, **44(1)**:1–29, 1997. 78, 84

[149] B. Mohar. A linear time algorithm for embedding graphs in an arbitrary surface. *SIAM Journal on Discrete Mathematics*, **12(1)**:6–26, 1999. 6

[150] B. Mohar, and C. Thomassen. *Graphs on Surfaces.* The Johns Hopkins University Press, 2001. 6

[151] B. Monien and E. Speckenmeyer. Ramsey numbers and an approximation algorithm for the vertex cover problem. *Acta Informatica*, **22**:115–123, 1985. 9

[152] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995. 1

[153] G. L. Nemhauser and L. E. Trotter. Vertex packing: structural properties and algorithms. *Mathematical Programming*, **8**:232–248, 1975. 20, 21

[154] J. v. Neumann and O. Morgenstern. *Theory of Games and Economic Behavior.* Princeton University Press, 3rd edition, 1953. 11

[155] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Habilitationsschrift, Universität Tübingen, Germany. 2002. 3

[156] R. Niedermeier and P. Rossmanith. Upper bounds for vertex cover further improved. In *Proceedings 16th STACS*, Springer-Verlag LNCS 1563, pages 561–570, 1999. 9, 43

[157] R. Niedermeier and P. Rossmanith. A general method to speed up fixed-parameter algorithms. *Information Processing Letters*, **73**:125–129, 2000. 19, 39

[158] R. Niedermeier and P. Rossmanith. An efficient fixed parameter algorithm for 3-Hitting Set. *Journal of Discrete Algorithms*, **2(1)**:93–107, 2002. 159

[159] T. Nishizeki and N. Chiba. Planar graphs: theory and applications. *Annals of Discrete Mathematics*, volume **32**, North-Holland, 1988. 4, 5, 127, 156

[160] A. Pagourtzis, P. Penna, K. Schlude, K. Steinhöfel, D. S. Taylor, and P. Widmayer. Server placements, roman domination, and other dominating set variants. In *Proceedings 2nd IFIP TCS (Foundations of Information Technology in the Era of Network and Mobile Computing)*, Kluwer Academic Publishers, pages 280–291, 2002. 11, 154

[161] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994. 6, 11

[162] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, **43**:425–440, 1991. 11, 77, 91

[163] V. T. Paschos. A survey of approximately optimal solutions to some covering and packing problems. *ACM Computing Surveys*, **29(2)**:171–209, 1997. 20

[164] A. Paz and S. Moran. Nondeterministic polynomial optimization problems and their approximations. *Theoretical Computer Science*, **15**:251–277, 1981. 12

[165] V. Raman. Parameterized complexity. In *Proceedings 7th National Seminar on Theoretical Computer Science (Chennai, India)*, pages I-1–I-18, June 1997. 3

[166] R. C. Read. Prospects for graph theory algorithms. *Annals of Discrete Mathematics*, **55**:201–210, 1993. 13, 120

[167] B. Reed. Finding approximate separators and computing tree-width quickly. In *Proceedings 24th ACM STOC*, pages 221–228, 1992. 97

[168] G. Ringel and J. W. T. Youngs. Solution of the Heawood map-coloring problem. Proc. Nat. Acad. Sci. U.S.A., **60**:438–445, 1968. 44

[169] F. S. Roberts. *Graph Theory and Its Applications to Problems of Society*. SIAM Press, 1978. Third printing, Odyssey Press, 1993. 11

[170] N. Robertson, D. P. Sanders, P. D. Seymour, and R. Thomas. Efficiently four-coloring planar graphs. In *Proceedings 28th ACM STOC*, pages 571–575, 1996. 20

[171] N. Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, **7**:309–322, 1986. 94

[172] N. Robertson and P. D. Seymour. Graph minors. X. Obstructions to tree-decomposition. *Journal of Combinatorial Theory, Series B*, **52**:153–190, 1991. 96, 155

[173] N. Robertson and P. D. Seymour. Excluding a graph with one crossing. *Graph structure theory (Seattle, WA, 1991)*, Amer. Math. Soc., Providence, RI, pages 669–675, 1993. 156

[174] J. M. Robson. Algorithms for maximum independent sets. *Journal of Algorithms*, **7(3)**:425–440, 1986. 91

[175] J. M. Robson. Finding a maximum independent set in time $O(2^{n/4})$. Technical Report 1251-01, Université Bordeaux, LaBRI, January 2001. 91

[176] P. D. Seymour and R. Thomas. Graph searching and a minmax theorem for treewidth. *Journal of Combinatorial Theory, Series B*, **58**:239–257, 1993. 95, 96

[177] P. D. Seymour and R. Thomas. Call routing and the ratcatcher. *Combinatorica*, **14**:217–241, 1994. 155

[178] H. G. Singh and R. P. Pargas. A parallel implementation for the domination number of grid graphs. *Congressus Numerantium*, **59**:297–312, 1987. 12

[179] D. A. Spielman and S.-H. Teng. Disk packings and planar separators. In *Proceedings 12th Annual ACM Symposium on Computational Geometry (SCG)*, pages 349–358, 1996. 65

[180] U. Stege and M. R. Fellows. An improved fixed-parameter-tractable algorithm for vertex cover. Technical Report 318, ETH Zürich, Department of Computer Science, April 1999. 9, 43

[181] J. A. Telle. Complexity of domination-type problems in graphs. *Nordic Journal of Computing* **1**:157–171, 1994. 13, 119

[182] J. A. Telle. Tree-decompositions of small pathwidth. Manuscript, 2002. To appear in *Discrete Applied Mathematics*. 154

[183] J. A. Telle and A. Proskurowski. Practical algorithms on partial k-trees with an application to domination-like problems. In *Proceedings 3rd WADS*, Springer-Verlag LNCS 709, pages 610–621, 1993. III, 12, 15, 93, 112, 113, 114, 119

[184] J. A. Telle and A. Proskurowski. Algorithms for vertex partitioning problems on partial k-trees. *SIAM Journal on Discrete Mathematics*, **10(4)**:529–550, 1997. III, 12, 15, 93, 112, 113, 114, 119

[185] W. T. Tutte. Convex representation of graphs. *Proc. London Math. Soc.*, **10(3)**:304–320, 1960. 79

[186] W. T. Tutte. How to draw a graph. *Proc. London Math. Soc.*, **13(3)**:743–768, 1963. 79

[187] S. M. Venkatesan. Improved constants for some separator theorems. *Journal of Algorithms*, **8**:572–578, 1987. 65

[188] K. Weihe. Covering trains by stations or the power of data reduction. In *Proceedings 1st ALEX*, pages 1–8, 1998. `http://rtm.science.unitn.it/alex98/proceedings.html`. 17

[189] K. Weihe. On the differences between "practical" and "applied" (invited paper). In *Proceedings 4th WAE*, Springer-Verlag LNCS 1982, pages 1–10, 2000. 17

[190] J. Wu and H. Li. Domination and its applications in ad hoc wireless networks with unidirectional links. In *Proceedings 29th ICPP*, IEEE Computer Society online publication, pages 189–200, 2000. 11

Lebens- und Bildungsgang

Name: Jochen Alber
Familienstand: verheiratet (seit 2.8.2002)

28.6.1973	geboren in Göppingen
1980 – 1984	Besuch der Grundschule in Ebersbach an der Fils
1984 – 1993	Besuch des Raichberg-Gymnasiums Ebersbach (RGE)
1991 – 1992	Schülersprecher am RGE
05/1993	Abitur (Note: 1,0) Leistungskurse: Mathematik und Physik
07/1993 – 10/1994	Zivildienst in der Geschwister-Scholl Jugendherberge in Ulm
10/1994 – 12/1999	Studium der Mathematik mit Nebenfach Informatik an der Eberhard-Karls-Universität Tübingen
10/1996	Vordiplom in Mathematik, Nebenfach Informatik (Note: 1,0)
07/1997	Studienarbeit (bei Prof. K.-J. Lange) im Fach Informatik (vgl. [13]): *"Lokalitätseigenschaften diskreter raumfüllender Kurven: Informatikrelevante Ergebnisse"*
08/1997 – 08/1998	Auslandsstudium an der University of Washington, Seattle, USA, im Rahmen des ISM (Internationaler Studiengang Mathematik)
08/1998 – 10/1998	Beschäftigung bei IBM Global Services in Böblingen im Projekt: *"Genetische Algorithmen zur Optimierung von Laserschweißprozessen"*
07/1999	Diplomarbeit (bei Prof. R. Nagel) im Fach Mathematik (vgl. [3]): *"Von der Struktur implementierter Halbgruppen mit Anwendungen in der Stabilitätstheorie"*
12/1999	Diplom in Mathematik, Nebenfach Informatik (Note: "mit Auszeichnung")
seit 02/2000	Promotion an der Fakultät für Informatik, Universität Tübingen, Lehrstuhl für Theoretische Informatik/Formale Sprachen (Prof. K.-J. Lange)
	Gutachter für: Journal of Computer and System Sciences, Information Processing Letters, STACS 2002, MFCS 2002, WG 2002, SODA 2003
	Publikationen: [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]