Exploiting structure to cope with NP-hard graph problems: Polynomial and exponential time exact algorithms

PIM VAN-'T-HOF

Exploiting structure to cope with NP-hard graph problems

Polynomial and exponential time exact algorithms

Pim van 't Hof

School of Engineering and Computing Sciences
Durham University

A thesis submitted for the degree of
Doctor of Philosophy

May 2010

Abstract

An ideal algorithm for solving a particular problem always finds an optimal solution, finds such a solution for every possible instance, and finds it in polynomial time. When dealing with NP-hard problems, algorithms can only be expected to possess at most two out of these three desirable properties. All algorithms presented in this thesis are *exact algorithms*, which means that they always find an optimal solution. Demanding the solution to be optimal means that other concessions have to be made when designing an exact algorithm for an NP-hard problem: we either have to impose restrictions on the instances of the problem in order to achieve a polynomial time complexity, or we have to abandon the requirement that the worst-case running time has to be polynomial. In some cases, when the problem under consideration remains NP-hard on restricted input, we are even forced to do both.

Most of the problems studied in this thesis deal with partitioning the vertex set of a given graph. In the other problems the task is to find certain types of paths and cycles in graphs. The problems all have in common that they are NP-hard on general graphs. We present several polynomial time algorithms for solving restrictions of these problems to specific graph classes, in particular graphs without long induced paths, chordal graphs and claw-free graphs. For problems that remain NP-hard even on restricted input we present exact exponential time algorithms. In the design of each of our algorithms, structural graph properties have been heavily exploited. Apart from using existing structural results, we prove new structural properties of certain types of graphs in order to obtain our algorithmic results.

Declaration

No part of this thesis has previously been submitted for any degree at any institution. Most of the results presented in this thesis have appeared, often in preliminary form, in the papers [50, 155, 156, 157, 158, 159, 160, 161], all of which have been subject to peer review. At the beginning of each chapter we mention where the results presented in that chapter have been published. Although many of the results have been obtained in collaboration, I have been heavily involved in and actively contributed to discussions that led to the results in every section of this thesis.

Acknowledgments

Table tennis is a great sport. Not only because it doesn't involve running around a muddy pitch in the freezing cold at six o'clock in the morning, but also because, very occasionally, table tennis brings together an undergraduate mathematics student and his future PhD supervisor. Many years passed between the moment Daniel Paulusma and I first met during a table tennis practice session at the University of Twente in Enschede, the Netherlands, and the moment I started as his PhD student at Durham University. Daniel, thank you for making me feel like a colleague rather than a student from the very first moment, for sharing me with the pupils of Consett Community Sports College for one year, for always finding time for me, and for teaching me so much during the many discussions we had. Could you please tell the people at EPSRC that their financial support is gratefully acknowledged?

A big thank you goes to Hajo Broersma, for bringing Daniel's PhD position to my attention while I was writing my Master's thesis in Klagenfurt, Austria, for acting as my second supervisor (again), and for giving me valuable advice whenever I needed it. Pinar Heggernes and Iain Stewart, thank you for agreeing to be my examiners, for carefully reading this thesis, and for making the viva such an interesting and pleasant occasion. Since working with others on nice problems is one of the most enjoyable aspects of life as a researcher, I also wish to express my sincere gratitude to all the coauthors of the papers that form the basis of this thesis: Hajo Broersma, Fedor Fomin, Marcin Kamiński, Daniël Paulusma, Johan van Rooij, Stefan Szeider, Dimitrios Thilikos, and Gerhard Woeginger.

During my time in Durham I have been fortunate enough to get to know many great people. I have made too many friends to mention all of them here. With the risk of offending most and losing some, I'll mention just one.

Mark Rhodes, what were the chances that one of my fellow PhD students would not only share my passion for mathematics, table tennis and pool, but would also become such a dear friend? (You can stop calculating now, Mark, the question was rhetorical.) Thank you for keeping me company on many occasions: you brighten up {m,usuall,alread,prett,sunn,da,ever}y time we meet, especially when you bring your lovely Laura and your incredibly adorable baby girl Annabelle.

Lotte and Hugh Shankland, thank you so much for allowing me to live in your amazing museum of a house, for making me feel at home in the wonderful little city of Durham, and for being so much more than landlords alone.

Back home, on the other side of the North Sea, there are a few people who deserve a special mention; having to miss their company on a regular basis is the main reason why, one day, I might return to the Netherlands. Susanne and Niels Besseling[1], thank you for being the great friends you are, and for always offering me to stay at your place whenever I visit Enschede. Mirjam en Stijn Nijenboer[2], thank you for being the great friends you are, and for always offering me to stay at your place whenever I visit Enschede. I can't wait to see your little man. Cees Brans, ome Cees, the mathematical exercises you gave me, or rather the Mars bars that formed the reward for solving them, contributed in no small part to my growing interest in mathematics; thank you for asking me why I didn't consider studying mathematics at university before I even realized that was an option. My brothers Koen and Jops[3], and Jops' soon-to-be-wife and even-sooner-to-be-doctor Nienke, you guys make me laugh out loud (lol) every time we speak, which is not nearly often enough. And finally, this acknowledgments section wouldn't be complete without mentioning the people I owe everything to. Lieve mama en papa, ik hou van jullie.

[1]It was a huge honor for me to be "ceremoniemeester" at your wedding.
[2]It was a huge honor for me to be "ceremoniemeester" at your wedding.
[3]It is a huge honor for me to be "ceremoniemeester" at your wedding.

Dedicated to my parents, Jeannette & Peter.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

One of the most well-known conjectures in theoretical computer science states that the class P of decision problems solvable in polynomial time by a deterministic Turing machine does not equal the class NP of decision problems solvable in polynomial time by a non-deterministic Turing machine. The validity of this conjecture would imply that we will never find an algorithm with polynomial worst-case running time that solves an NP-hard problem. Since the $\mathsf{P} \neq \mathsf{NP}$ conjecture is widely believed to be true and numerous interesting computational problems have been shown to be NP-hard, a lot of research is devoted to finding ways to cope with the intrinsic hardness of such problems. Ideally, we would like an algorithm to find an optimal solution, find such a solution for every possible instance, and find it in polynomial time. From the above it is clear that algorithms for solving NP-hard problems can only be expected to possess at most two out of these three desirable properties.

If the restriction that the obtained solution be optimal is dropped, then we find ourselves in the field of heuristics and approximation algorithms. A heuristic is a method for solving a problem without any guarantee that the obtained solution is close to optimal. In fact, it is often possible to create instances on which a heuristic can be shown to perform poorly. However, despite the lack of any theoretical guarantee on their performance, heuristics are widely used as they typically work very well on many inputs in practice (see for example [27, 133]). Unlike a heuristic, an approximation algorithm produces provably good, albeit suboptimal, solutions. Traditionally, approximation algorithms run in polynomial time and find solutions that are within

a reasonable factor of the optimal solution. Unfortunately, for several important optimization problems it has been shown that they are NP-hard to approximate within a non-trivial factor (see for example [270]). Motivated by these discouraging inapproximability results, approximation algorithms with exponential worst-case running times have recently been proposed (see for example [43, 86, 122]). These algorithms show that researchers are sometimes willing to drop not just one, but two out of the three aforementioned desirable properties when dealing with certain NP-hard problems.

All algorithms presented in this thesis are *exact algorithms*, which means that they solve problems exactly by always finding an optimal solution. Demanding the solution to be optimal means that other concessions have to be made when designing an exact algorithm: we either have to impose restrictions on the instances of the problem in order to achieve a polynomial time complexity, or we have to abandon the requirement that the worst-case running time has to be polynomial. In some cases, for example in the algorithms presented in Chapters 3 and 7 of this thesis, we are even forced to do both.

Often instances of an NP-hard problem arising in practice may be assumed to have a certain structure, and sometimes this structure can be used to solve the problem efficiently. Apart from this practical motivation, it is interesting from a theoretical point of view to identify classes of instances for which an NP-hard problem can be solved in polynomial time, as this might provide insight into what makes certain problems hard. In Section 1.2 we further motivate the study of polynomial time algorithms on restricted input and survey some relevant results in the literature. In particular, we focus on algorithmic results obtained on NP-hard problems restricted to three specific graph classes, as for most NP-hard problems studied in this thesis we present algorithms for instances belonging to one of these three classes.

If the problem under consideration remains NP-hard even for restricted input, or if we require an exact algorithm for solving an NP-hard problem on general instances, then we have to settle for a super-polynomial time complexity. Clearly, a trivial brute force algorithm that solves an NP-hard problem by enumerating and checking all possible solutions is an example of such an algorithm. Often though it is possible to find an exact algorithm that has a better worst-case running time than the trivial algorithm, and it is in those "moderately exponential time algorithms" that we are interested.

In Section 1.3 we argue why the study of exact exponential time algorithms has quickly established itself as a popular field within theoretical computer science, and briefly describe some techniques that have successfully been used in the design and analysis of such algorithms.

Before taking a closer look at polynomial and exponential time exact algorithms in Section 1.2 and Section 1.3 respectively, we start by introducing the necessary notation and terminology in Section 1.1. We conclude this chapter by giving a brief overview of the rest of this thesis in Section 1.4.

1.1 Notation and terminology

Most of the standard graph theoretic terminology presented below is derived from the book by Diestel [91], and we refer to that book for graph terminology not defined below. Several definitions will be given later in the thesis, either because they are used in only one chapter of this thesis, or because the preceding theory provides insight into the concepts in question and makes their definitions appear more naturally. For a self-proclaimed "relatively low-level introduction to some of the central notions of computational complexity", in particular focussing on the theory of NP-completeness, we refer to the classic text book by Garey and Johnson [127]. Papadimitriou [219] gives a very thorough and formal introduction to computational complexity theory. The $\mathcal{O}^*$-notation, used throughout the thesis to denote the time complexity of an exponential time algorithm, indicates that we suppress polynomially bounded factors. In other words, for any exponential function f, we have

$$\mathcal{O}^*(f(n)) = \mathcal{O}(f(n) \cdot n^{\mathcal{O}(1)}) \ .$$

1.1.1 The basics

A *graph* $G = (V, E)$ is an ordered pair of finite sets, where V is a non-empty set whose elements are called the *vertices* of G, and E is a set of unordered pairs (u, v) with $u, v \in V$ called the *edges* of G. We refer to the vertex set and edge set of a graph G as $V(G)$ and $E(G)$, respectively, in case the names of these sets are not explicitly specified. Throughout this thesis, we use n and m to denote the number of vertices and edges of the graph under consideration, respectively. In particular, whenever we deal with algorithms,

n and m always refer to the number of vertices and edges of the *input* graph. The vertices u and v are called the *end vertices* of the edge (u, v). Apart from some of the so-called role graphs that will be considered in Chapter 5, none of the graphs in this thesis have *loops*, i.e., edges of the form (u, u). Instead of (u, v) we will consistently write uv to represent an edge between u and v.

The (open) *neighborhood* of a vertex v in G is the set $N_G(v) = \{y \in V(G) \mid xy \in E(G)\}$ of neighbors of v in G. The *closed neighborhood* of v is the set $N_G[v] = N_G(v) \cup \{u\}$. For any set $S \subseteq V(G)$, we write $N_S(v) = N_G(v) \cap S$ and $N_G(S) = \cup_{u \in S} N_G(u) \setminus S$. The *degree* of v in G, denoted $d_G(v)$, is the number of neighbors of v in G, i.e., $d_G(v) = |N_G(v)|$. A vertex of degree 0 is called *isolated*, and a vertex of degree 1 is called *pendant*. If $N_G(v) = V(G) \setminus \{v\}$ for every vertex v, then G is called *complete*. We use K_k to denote the complete graph on k vertices. A vertex is called *simplicial* if its neighborhood induces as complete graph. If no confusion is possible, we write $d(v)$, $N(v)$, and $N(S)$ instead of $d_G(v)$, $N_G(v)$, and $N_G(S)$, respectively.

We say that a graph G' is a *subgraph* of a graph G if $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$. If G' is a subgraph of G and $V(G') = V(G)$, we say that G' is a *spanning* subgraph of G. A subgraph G' of G is an *induced subgraph* of G if for every pair of vertices $u, v \in V(G')$ we have $uv \in E(G')$ if and only if $uv \in E(G)$; we say that $V(G')$ *induces* the subgraph G'. For any non-empty set $S \subseteq V$, we write $G[S]$ to denote the subgraph of G induced by S. We say that the set S is *connected* if $G[S]$ is connected. A vertex v is *adjacent* to a connected set S if v has at least one neighbor in S. Two connected sets $S_1, S_2 \subseteq V(G)$ are *adjacent* if S_1 contains at least one vertex that is adjacent to S_2. For any proper subset $S \subset V(G)$, we write $G - S$ to denote the graph $G[V(G) \setminus S]$, i.e., the graph obtained from G by removing all vertices of S and their incident edges. If $S = \{v\}$, we write $G - v$ instead of $G - \{v\}$. Similarly, for any set $F \subseteq E(G)$, the graph $G - F$ is the graph obtained from G by removing all edges of F. A *separator* of a connected graph G is a set S of vertices of G such that $G - S$ is not connected. If a separator S of G consists of a single vertex s, then s is called a *cut vertex* of G.

A subset $S \subseteq V$ is called a *clique* if $G[S]$ is a complete graph, and S is called an *independent set* if $G[S]$ contains no edges. A clique S is called *maximal* if for every proper superset S' of S the graph $G[S']$ is not complete,

and S is called *maximum* if G has no clique containing strictly more vertices than S; maximal and maximum independent sets are defined analogously. A set $U \subseteq V(G)$ *dominates* a set $U' \subseteq V(G)$ if every vertex $v \in U'$ either belongs to U or has a neighbor in U. We also say that U dominates the graph $G[U']$. A subgraph H of G is a *dominating subgraph* of G if $V(H)$ dominates G. A *dominating* vertex is a vertex adjacent to all other vertices.

We write P_k and C_k to denote the chordless path and the chordless cycle on k vertices, respectively. The *length* of a path or a cycle refers to its number of edges. Note that the path P_k has length $k-1$. A path or a cycle is said to be *odd* (respectively *even*) if it has odd (respectively even) length. A *hole* in a graph G is an induced subgraph of G that is a chordless cycle of length at least 4, and an *antihole* is the complement of a hole. The length of an antihole is defined to be the length of its complement, and an antihole is called odd (respectively even) if its complement is an odd (respectively even) hole.

A *hypergraph* H is a pair $(Q, \mathcal{S})$ consisting of a set $Q = \{q_1, \ldots, q_m\}$, called the *vertices* of H, and a set $\mathcal{S} = \{S_1, \ldots, S_n\}$ of non-empty subsets of Q, called the *hyperedges* of H. For any $S \in \mathcal{S}$, we write $H - S$ to denote the hypergraph $(Q, \mathcal{S} \setminus S)$. With a hypergraph $H = (Q, \mathcal{S})$ we associate its *incidence graph* I, which is a bipartite graph with partition classes Q and $\mathcal{S}$, where for any $q \in Q, S \in \mathcal{S}$ we have $qS \in E(I)$ if and only if $q \in S$. Note that a graph is a hypergraph for which every hyperedge contains exactly two vertices, which means that hypergraphs can be seen as a generalization of graphs. A 2-*coloring* of a hypergraph $H = (Q, \mathcal{S})$ is a partition (Q_1, Q_2) of Q such that $Q_1 \cap S_j \neq \emptyset$ and $Q_2 \cap S_j \neq \emptyset$ for $1 \leq j \leq n$.

1.1.2 Some graph classes

A graph G is called *bipartite* if $V(G)$ can be partitioned into two independent sets A and B, called the *partition classes* of G. A bipartite graph G with partition classes A and B is called *complete* if every vertex of A is adjacent to every vertex of B. We write $K_{\ell,m}$ to denote a complete bipartite graph whose two partition classes contain ℓ and m vertices, respectively. A *star* is a complete bipartite graph one partition class of which contains exactly one vertex. The unique vertex in this class is called the *center* of the star. A *claw* is a four-vertex star $K_{1,3} = (\{x, a, b, c\}, \{xa, xb, xc\})$, and vertex x is called

the *center* of the claw.

A graph G is called *H-free* for some graph H if G does not contain an induced subgraph isomorphic to H. In particular, a graph is called *triangle-free* if it does not contain a subgraph isomorphic to the cycle on three vertices, and a graph is called *claw-free* if it has no induced subgraph isomorphic to the *claw*. For any family $\mathcal{H}$ of graphs, we write $Forb(\mathcal{H})$ to denote the class of graphs that are H-free for every $H \in \mathcal{H}$.

The *line graph* of a graph H with edges $e_1, \ldots, e_p$ is the graph $L(H)$ with vertices $u_1, \ldots, u_p$ such that there is an edge between any two vertices u_i and u_j if and only if e_i and e_j share one end vertex in H. Note that $L(K_3) = L(K_{1,3}) = K_3$; it is well-known that every connected line graph $F \neq K_3$ has a unique H with $F = L(H)$ (see for example [145]). We call H the *preimage graph* of F. For K_3 we let $K_{1,3}$ be its preimage graph.

Let $\mathcal{F}$ be a family of non-empty sets. The *intersection graph* of $\mathcal{F}$ is obtained by representing each set in $\mathcal{F}$ by a vertex and connecting two vertices if and only if the two corresponding sets intersect. The intersection graph of a family of intervals on the real line is called an *interval graph*. If none of the intervals is properly contained in another interval, then we have a *proper interval graph*. A graph is a proper interval graph if and only if it is a *unit interval graph*, i.e., the intersection graph of a family of unit intervals on the real line [229]. Roberts [229] also showed that proper interval graphs are exactly the claw-free interval graphs. Proper interval graphs are also known as *indifference graphs* [229].

A graph G is *perfect* if for every induced subgraph H the chromatic number of H equals the size of a largest clique in H. A graph is *chordal* if it does not contain a hole, i.e., an induced cycle of length at least 4. A *split graph* is a graph whose vertex set can be partitioned into a clique and an independent set. A graph G is a split graph if and only if both G and its complement $\overline{G}$ are chordal [111].

1.1.3 Minors, induced minors and contractions

Let G and H be two graphs. The *edge contraction* of edge uv in G removes u and v from G, and replaces them by a new vertex adjacent to precisely those vertices that were adjacent to $\{u, v\}$ in G. We denote the resulting graph by $G \backslash uv$. If H can be obtained from G by a sequence of edge contractions, vertex

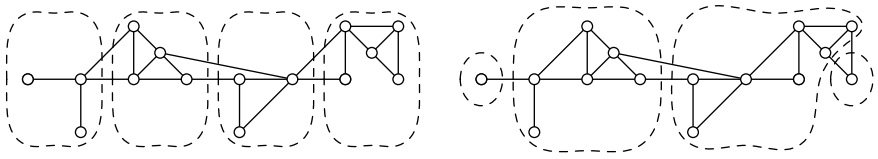

Figure 1.1: Two P_4-witness structures of a graph.

deletions and edge deletions, then H is a *minor* of G. If H can be obtained from G by a sequence of edge contractions and vertex deletions, then H is an *induced minor* of G. If H can be obtained from G by a sequence of edge contractions, then H is a *contraction* of G, and G is called *H-contractible*.

By definition, G is H-contractible if and only if G has a so-called *H-witness structure* $\mathcal{W}$, which is a partition of $V(G)$ into $|V(H)|$ non-empty connected sets, called *H-witness sets*, such that every vertex $h \in V(H)$ corresponds to an H-witness set $W(h)$ of G, and for every two vertices $h_i, h_j \in V(H)$, witness sets $W(h_i)$ and $W(h_j)$ are adjacent in G if and only if h_i and h_j are adjacent in H. By contracting all the edges in each of the witness sets, we obtain the graph H. See Figure 1.1 for an example that shows that, in general, a witness structure $\mathcal{W}$ is not uniquely defined. In both witness structures shown in Figure 1.1, contracting all the edges in each of the witness sets results in the graph P_4.

1.2 Polynomial time algorithms on restricted input

Ever since Berge coined the term "perfect graph" in 1963 [25], the class of perfect graphs has gone on to become one of the most intensively studied classes in graph theory. In structural graph theory, one of the most challenging tasks was to prove Berge's Strong Perfect Graph Conjecture, stating that a graph is perfect if and only if it does not contain an odd hole or an odd antihole as an induced subgraph. The quest for an affirmative answer to this question, eventually obtained by Chudnovsky et al. [64], would take over 40 years, and culminated in one of the most eagerly anticipated results in graph theory. Along the way, not only structural but also many algorithmic results were obtained for specific subclasses of perfect graphs. This led Golumbic [137] to write a book, entitled "Algorithmic Graph Theory and

Perfect Graphs", in an attempt to "collect and unify the topic to act as a springboard for researchers, and especially graduate students, to pursue new directions of investigations".

In part driven by this call, many researchers continued to work on perfect graphs. Entire books have been devoted to particular families of perfect graphs, such as tolerance graphs [138] and even the very restricted class of threshold graphs [206], which is exactly the intersection of cographs and split graphs. In the foreword of the second edition of his book, published in 2004, Golumbic [137] states that "the world of perfect graphs has grown to include over 200 special graph classes". Coincidentally, Brandstädt, Le and Spinrad [47] mention in the foreword of their 1999 book "Graph Classes: A Survey" that they describe "almost 200 classes". In contrast to Golumbic's book however, the graph classes included in the book of Brandtädt et al. are not all subclasses of perfect graphs. The same holds for the numerous classes that make an appearance in the monograph "Efficient Graph Representations" by Spinrad [251]. This shows that the study of graphs with particular structure has most certainly transcended the world of perfect graphs, even though it was this class that originally caused this field to flourish.

We will not cover 200 graph classes in this section. Instead, we have chosen to restrict our attention here to three graph classes that play an important role in this thesis, as almost all algorithms presented here have been designed to process graphs that belong to one of these classes. We focus on graphs without long induced paths, chordal graphs and claw-free graphs in Sections 1.2.2, 1.2.3 and 1.2.4, respectively, as they will appear in that order in Chapters 2 to 7. In an effort to motivate the study of these particular graph classes, we list several structural properties and algorithmic results that have been obtained for these classes. It is not our intention to give a complete overview of all the results known for these particular graph classes; the examples included here are merely illustrative. We start in Section 1.2.1 by motivating the study of restrictions of NP-hard problems to special graph classes, although one could argue that the publication of the five aforementioned books on the topic renders this unnecessary.

1.2.1 Why restricted input?

The most natural motivation for studying restrictions of an NP-hard problem to specific graph classes is the fact that sometimes those graph classes simply present themselves in practical applications of the problem. Representing countries by a vertex and joining vertices if and only if the corresponding countries share a border results in a planar graph. The problem of determining how many colors are needed to color the countries in such a way that no two bordering countries receive the same color is then equivalent to solving the Chromatic Number problem on planar graphs. In this case, it might even be argued that solving the Chromatic Number problem on general graphs is less natural than solving the problem on planar graphs. There are also very natural applications of Chromatic Number and related coloring problems on restricted input in the field of scheduling (see for example [137, 207]). For example, the problem of assigning k aircrafts to n flights, where the ith flight has to be scheduled in the time interval (a_i, b_i) and no aircraft can be assigned to two flights with overlapping time intervals, can be modeled as a coloring problem on interval graphs. Assigning the minimum number of frequencies to radio stations in such a way that interference is prevented is equivalent to solving the Chromatic Number problem on unit disk graphs. It is clear that for coloring problems alone numerous applications exist in which the input graphs may be assumed to belong to a specific graph class, and these form just the tip of the iceberg.

Sometimes structure shows up rather more unexpectedly. Many algorithms for solving problems on general graphs start by preprocessing the input graph, after which several reduction and branching rules are executed. This results in a "core" graph for which the problem has to be solved. These core graphs, sometimes called kernels, often have a particular structure, which can be exploited in order to design a relatively fast algorithm for solving the problem on this type of graph [95]. Using this fast algorithm as a subroutine in the main algorithm might speed up the overall running time. The problems on these structured core graphs can be solved using ad-hoc methods or by combining known results on this type of graphs (see for example [2, 174, 259]). A more thorough study of algorithms on restricted input might provide new tools that can be used by researchers in different ar-

eas having to deal with specific graph classes. It is important to note that it is often hard to predict what kind of structure lies at the heart of a problem. After all, the type of graph that remains after the first stages of an algorithm depends on the choices the researchers make whilst designing the algorithm. For this reason one might argue that it is not always necessary to justify in advance why a certain graph class is studied, as the knowledge gained from studying that graph class might one day come in handy, when this particular graph class presents itself in the final stages of an exact algorithm on general input.

The following question can be seen as another motivation for studying the algorithmic behavior of NP-hard problems on certain types of graphs: how far, and in what way, do we need to restrict the input before a certain NP-hard problem can be solved in polynomial time? A lot of research is devoted to investigating the boundary between polynomial time solvable and NP-hard cases of certain problems. Attempts to narrow the gap between easy and hard cases can start "from above" or "from below": one can either try to identify smaller classes of graphs on which the problem remains NP-hard, or one can try to extend polynomial time results on a specific graph class to one of its superclasses. This sometimes leads to the publication of a table in which all polynomial and NP-hard cases of a certain family of problems are summarized [266]. And then the race is on! The table reappears in every subsequent paper on the topic, each time with one or more new results added [53, 152, 190, 226]. Every empty entry yields a straightforward open question, so researchers are encouraged to join in the game of narrowing the gap in order to complete the table. We will discuss one of those tables in Section 1.2.2. If an NP-hard problem turns out to be polynomial time solvable on some graph classes, but remains NP-hard on others, then we might get an idea of what it is that makes this specific problem hard. Also knowing that some NP-hard problems behave similarly on special graph classes might uncover some connection between those problems that might otherwise have been overlooked.

It is not always the input graph where structure can be found. Sometimes a problem imposes a certain structure on the *output*. A good example is the family of editing problems, asking whether a certain type of target graph can be obtained from the input graph by adding and/or removing a cer-

tain number of vertices and/or edges. Editing problems have applications in problems related to DNA physical mapping [261] and sparse matrix computations [175]. Common graph classes that appear in this context are chordal graphs, interval graphs and cluster graphs. Cluster graphs, also known as P_3-free graphs, will be briefly mentioned in Section 1.2.2, and chordal graphs return in Section 1.2.3. Typically, an editing problem prescribes a graph *class* to which the output graph must belong, and sometimes extra restrictions on the output are added. It is also possible that the exact target graph is prescribed. The two problems studied in Chapter 4 are of this type, as they ask if a certain input graph G can be transformed into a fixed target graph H by performing edge contractions and vertex deletions. Most of the results obtained in that chapter heavily rely on structural properties of the target graph H.

1.2.2 P_k-free graphs

To get the trivial cases out of the way, let us observe that P_1-free graphs do not exist due to the fact that we defined a graph to have a non-empty vertex set, and that a graph is P_2-free if and only if it has no edges. The first class of real interest is the class of P_3-free graphs. It is easy to see that a graph is P_3-free if and only if each of its connected components is a complete graph. For this reason, P_3-free graphs are also called cluster graphs. Cluster graphs have applications in several fields where large sets of data need to be "clustered", such as computational biology [248], image processing [269] and VLSI design [144]. Shamir et al. [247] describe in detail how such real life applications can modeled using cluster graphs, and they prove a range of results involving cluster editing problems. Next we arrive at the class of P_4-free graphs. This class, better known as the class of cographs, is without doubt the most intensively studied class in the family of P_k-free graphs. The class of cographs was discovered independently by several authors in the 1970s [191, 246, 253]. Many of the early results, including eight equivalent characterizations, are collected and extended in [82] (see also Theorem 11.3.3 in [47]). Worth mentioning is the fact that cographs have a unique tree representation, a so-called cotree [82]. This cotree representation has been used to obtain a linear time recognition algorithm for cographs [83], as well as polynomial time algorithms for restrictions of many NP-hard problems

such as Maximum Clique and Hamiltonian Cycle to cographs [82].

In contrast to cluster graphs and cographs, no special name seems to be reserved for the class of P_k-free graphs for any value $k \geq 5$. This did not stop several groups of authors from studying restrictions of NP-hard problems to these classes, especially over the past decade, as we will see below. A possible reason for this is the interest of researchers in investigating the boundary between polynomial time solvable and NP-hard cases of a problem, as mentioned in Section 1.2.1. The advantage of studying NP-hard problems on P_k-free graphs is the fact that the class of P_k-free graphs is a subclass of the class of P_m-free graphs if $k \leq m$. As a result, if a problem is shown to be NP-hard for the class of P_k-free graphs, then this also holds for the class of P_m-free graphs, for any $m \geq k$. Similarly, if a problem is polynomial time solvable on P_k-free graphs, then this is also true for the class of P_j-free graphs, for any $j \leq k$. Hence, in order to find a computational complexity classification of an NP-hard problem restricted to the class of P_k-free graphs for every k, one NP-hardness proof and one polynomial time algorithm suffice. We will present such complexity classifications for the two problems studied in Chapter 3.

The ℓ-Colorability problem is to determine whether the vertices of a given graph can be properly colored using at most ℓ colors. Over the past decade, restrictions of this problem to the class of P_k-free graphs have been studied by several groups of authors in an attempt to determine the computational complexity of the ℓ-Colorability problem for P_k-free graphs for various combinations of ℓ and k. This line of research was initiated in 2001 by Woeginger and Sgall [266], who proved NP-completeness of deciding whether a P_8-free graph is 5-colorable and of deciding whether a P_{12}-free graph is 4-colorable. They also presented a polynomial time algorithm for deciding whether a P_5-free graph can be 3-colored. Their paper is the first one to include a table, in which all the known complexity results of this type were summarized. The results obtained in [266] were improved and extended by several authors in subsequent years [49, 53, 152, 190, 226]. A big step forward was made when Hoàng et al. [152] presented a polynomial time algorithm for solving the ℓ-Colorability problem on P_5-free graphs for every value of ℓ.

The class of P_5-free graphs plays an interesting role with respect to the Maximum Independent Set problem, which is the problem of finding a

maximum size independent set in a graph: it is the only minimal graph class characterized by a single connected forbidden induced subgraph for which the computational complexity of this problem is unknown [135]. In an attempt to resolve this problem, a vast number of polynomial time algorithms for finding a maximum independent set on subclasses of P_5-free graphs have been proposed [8, 41, 48, 119, 134, 135, 202, 203, 210, 211, 212, 213]. Since the computational complexity of the Maximum Independent Set problem is also unknown for the class of P_6-free graphs, similar results have been obtained on subclasses of P_6-free graphs [45, 211, 214]. Very recently, Maffray [204] presented a polynomial time algorithm that, for every fixed ℓ, finds a maximum independent set in any ℓ-colorable P_5-free graph. His algorithm is based on a structural property which ensures that every connected ℓ-colorable P_5-free graph has a vertex whose non-neighbors induce a $(\ell - 1)$-colorable subgraph. We also mention two problems that can be solved in polynomial time on P_4-free graphs, but remain NP-hard on P_5-free graphs, and therefore also on P_k-free graphs for $k \geq 6$: determining the chromatic number [183], and finding a minimum dominating set (which remains NP-hard even on split graphs [29], a subclass of P_5-free graphs).

All the results mentioned in this section have been obtained by making use of structural properties of P_k-free graphs. Characterizations of P_k-free graphs that have proved to be particularly interesting involve connected dominating subgraphs of P_k-free graphs. For example, the algorithms in [53, 152] rely on a structural result due to Bacsó and Tuza [15], which states that every P_5-free graph has a dominating clique or a dominating induced P_3. More detailed information on characterizations of P_k-free graphs in terms of connected dominating subgraphs can be found in Chapter 2. In that chapter, we also present a new characterization of this type for the class of P_6-free graphs. We use this characterization in Chapter 3 to guarantee the relevant time complexities of the exact exponential time algorithms presented there.

1.2.3 Chordal graphs

The fact that Golumbic [137] devotes an entire chapter to chordal graphs, and Brandstädt et al. [47] even choose to introduce chordal graphs in the very first chapter, entitled "Basic concepts", of their book on graph classes, can be seen as evidence of the huge popularity of this class. One reason for this popularity

is the fact that chordal graphs have applications in many different fields, which means that they have been studied from different angles. The close link between chordal graphs and the problem of solving sparse symmetric systems of linear equations, first established by Parter [221] in 1961, has been well-documented [133]. The class of chordal graphs, and in some cases its proper subclass of interval graphs in particular, also plays a central role in database management systems [19, 256], computer vision [75], phylogenetic trees [55, 56], and many more areas [137]. The wide variety of applications also explains why chordal graphs appear under many different names in the literature, such as triangulated graphs, rigid-circuit graphs, monotone transitive graphs and perfect elimination graphs [47].

It is not only their appearance in various practical applications that make the class of chordal graphs one of the most well-studied graph classes in algorithmic graph theory. Chordal graphs have many interesting and diverse structural properties, which have been used successfully in the design of fast algorithms and are worth investigating from a theoretical viewpoint alone. One of the most celebrated characterizations of chordal graphs is based on a result by Dirac [92], stating that every chordal graph is either complete or has at least two non-adjacent simplicial vertices. Fulkerson and Gross [121] proved that this fact can be used in order to recognize chordal graphs in polynomial time: a simple algorithm that repeatedly finds and removes a simplicial vertex until no simplicial vertex remains will remove every vertex of the input graph if and only if that graph is chordal. If a graph is chordal, then the order in which this algorithm picks the vertices is called a perfect elimination ordering. The characterization due to Fulkerson and Gross [121] has therefore become widely known as "a graph is chordal if and only if it has a perfect elimination ordering", even though the words "perfect elimination" do not appear in their paper.

Perfect elimination orderings have proved to be a useful tool in developing fast algorithms for solving problems involving chordal graphs. Gavril [129] uses a perfect elimination ordering to find a maximum independent set of a chordal graph in polynomial time. Rose, Tarjan and Lueker [236] show that a perfect elimination ordering can be found in linear time, which implies that chordal graphs can be recognized in linear time. Perfect elimination orderings can also be linked to an algorithm called Elimination Game, first

described by Parter [221] in relation to Gaussian elimination on sparse symmetric matrices. We will take a close look at this game in Section 6.4.2, as it is used as a subroutine in an algorithm by Tarjan to find a clique separator decomposition of a graph. Tarjan's algorithm in turn appears as a subroutine in our algorithm for finding shortest induced paths of given parity between two specified vertices in a claw-free perfect graph, presented in Section 6.4.3.

Chordal graphs can been seen as a generalization of trees, as they are exactly the intersection graphs of subtrees of a tree [56, 130, 260]. Another characterization is due to Dirac [92], who showed that a graph is chordal if and only if every minimal separator is a clique. Perhaps even more important, especially from an algorithmic point of view, is the fact that the number of maximal cliques and the number of minimal separators in a chordal graph does not exceed the number of vertices of the graph [121], whereas an arbitrary graph might have an exponential number of maximal cliques and minimal separators. Since it is possible to find all the maximal cliques of a chordal graph in linear time [35, 137, 236], the Maximum Clique problem is solvable in linear time on chordal graphs. The same holds for the Chromatic Number problem, as chordal graphs were among the first classes of graphs shown to be perfect [137]. A particularly useful tool for representing the maximal cliques and minimal separators of a chordal graph is its so-called clique tree. We will return to clique trees in Section 5.2.1.

Researchers studying chordal graphs have a wealth of structural results at their disposal, which has led to the discovery of polynomial time algorithms on chordal graphs for many NP-hard problems. Despite the nice properties of chordal graphs, some problems remain NP-hard on this class. Finding a hamiltonian cycle is NP-hard on chordal graphs [30], and the same holds for the problem of finding a minimum dominating set [42]. In fact, both problems remain NP-hard when restricted to the class of path graphs, which are exactly the intersection graphs of paths in a tree and therefore form a subclass of chordal graphs. Two other examples are the notoriously hard problems Bandwidth and Cutwidth, each asking for a certain optimum linear ordering of the vertices of the input graph: these problems remain NP-hard even on trees of maximum degree 3 [126] and split graphs [150], respectively. On the positive side, the Bandwidth problem can be solved in polynomial time on interval graphs [181]. Without defining here what a

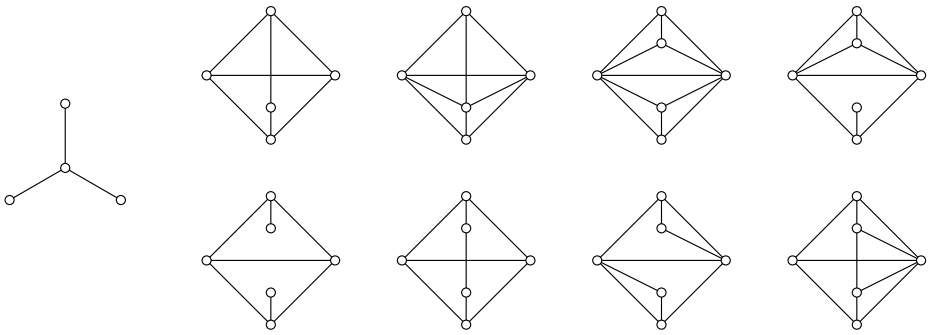

Figure 1.2: Beineke's nine forbidden induced subgraphs of a line graph.

k-role assignment is, we mention that in Chapter 5 we present a polynomial time algorithm for finding a 2-role assignment of a chordal graph, and we show that the problem of finding a k-role assignment is NP-hard on chordal graphs if $k \geq 3$.

1.2.4 Claw-free graphs

In order to properly introduce the class of claw-free graphs, let us first consider the related class of line graphs. Although the name "line graph" first appeared in a paper by Harary and Norman [147] in 1960, the concept of line graphs has been studied as early as 1932 [263]. Beineke [24] proved that a graph is a line graph if and only if it does not contain any of the nine graphs in Figure 1.2 as an induced subgraph. The smallest and simplest of these minimal forbidden induced subgraphs is the claw, which immediately implies that the class of claw-free graphs is a superclass of line graphs. Many NP-hard problems have been shown to be solvable in polynomial time for the class of line graphs, and it is natural to ask if these algorithmic results can be generalized to the class of claw-free graphs. Although Beineke's characterization of line graphs can thus be seen as the original motivation for the study of claw-free graphs, interest in the class was boosted in the 1970s and 1980s by several results related to perfect matchings [252, 188], hamiltonian cycles [139, 140] and perfect graphs [76, 222]. This increase in popularity led to the publication of hundreds of papers on the topic, almost 200 of which are referred to in a survey by Faudree, Flandrin and Ryjáček [102]. Although the

emphasis in [102] is on structural properties of claw-free graphs, the survey also contains a chapter on algorithmic results. We briefly mention a few of those results here, and refer to the extensive survey for many more results and background information.

A celebrated result by Edmonds [96] states that finding a maximum matching in a graph can be done in polynomial time. Since a maximum matching in a graph G corresponds to a maximum independent set in the line graph $L(G)$ of G, the Maximum Independent Set problem can be solved in polynomial time for line graphs. Using ideas from [96], Sbihi [241] managed to design a polynomial time algorithm for finding a maximum independent set in a claw-free graph. The following two results brought the class of claw-free graphs to the attention of the active perfect graph community and undoubtedly sparked a lot of interest in claw-free graphs. Parthasarathy and Ravindra [222] showed that Berge's Strong Perfect Graph Conjecture (see Section 6.2) holds for claw-free graphs, and Chvátal and Sbihi [76] presented a polynomial time algorithm for testing whether or not a claw-free graph is perfect. We will discuss this recognition algorithm in detail in Section 6.2, as we will use it as a subroutine in one of our algorithms presented in Chapter 6.

Despite the discovery of many polynomial time algorithms for claw-free graphs, numerous well-known algorithmic problems remain NP-hard on claw-free graphs. This automatically holds for problems that were shown to be NP-hard on line graphs. Examples of such problems are the Hamiltonian Cycle problem [28], and the Chromatic Number problem (an immediate corollary of the proof that the problem of determining the minimum number of colors needed to properly color the *edges* of a graph is NP-hard [163]). Since the Maximum Independent Set problem is NP-hard on triangle-free graphs, and the class of claw-free graphs contains the class of complements of triangle-free graphs, the problem of finding a maximum clique in a claw-free graph is NP-hard [225]. For several problems that are NP-hard for claw-free graphs it has been shown that they become polynomial time solvable when restricted to claw-free perfect graphs. For example, this is the case for the problems of determining the chromatic number [165] and finding a maximum clique [167]. In Section 6.4, we present a polynomial time algorithm for finding a shortest induced path of given parity between two given vertices in

a claw-free perfect graph.

Very recently, in a series of seven papers spanning over 200 pages in total, Chudnovsky and Seymour [66, 67, 68, 69, 70, 71, 72] proved a complete structure theorem for claw-free graphs. They identify a few basic types of claw-free graphs, and show that every connected claw-free graph can be obtained from one of those basic graphs by simple expansion operations. Even before the first paper in the series appeared, Chudnovsky and Seymour [65] already published a survey paper in which they give an exact statement of the theorem, sketch the proof and give some applications. The class of quasi-line graphs plays an important role in their papers. A graph is a *quasi-line graph* if the neighborhood of every vertex can be covered by two cliques. It is not hard to see that every line graph is a quasi-line graph, and that every quasi-line graph is claw-free. Therefore, if the solution to a problem is known for line graphs, solving the problem on quasi-line graphs can be an intermediate step in order to solve the problem on claw-free graphs. This was shown by King and Reed [180], who used the structural properties of quasi-line graphs described in [65] to prove the validity of a conjecture by Reed [227] on quasi-line graphs; the conjecture was known to be true for line graphs, but it is unclear if the conjecture holds for claw-free graphs. The paper by King and Reed inspired Fiala et al. [104], who study the k-IN-A-PATH problem of deciding whether a given graph has an induced path containing k specified vertices. This problem is known to be NP-complete on general graphs for any fixed $k \geq 3$ [90]. Fiala et al. [104] show that this problem is NP-complete on line graphs if k is part of the input, but can be solved in polynomial time on claw-free graphs for any fixed k. Their algorithm reduces the original problem on claw-free graphs to a problem on quasi-line graphs, and solves the problem using a characterization of quasi-line graphs from [65] and related algorithmic results from [180].

1.3 Exact exponential time algorithms

As we mentioned at the beginning of this chapter, exhaustively enumerating and checking all possible solutions allows us to trivially solve every NP-hard optimization problem, whose decision variant is NP-complete, in exponential time. It seems relatively useless trying to find faster exact algorithms

for NP-hard problems, since any such algorithm will still be an exponential time algorithm (unless, of course, P = NP) and therefore, at least from a theoretical point of view, hopelessly inefficient. Despite this, many exact exponential time algorithms have been proposed in the literature. Undoubtedly one of the most famous ones dates back to 1962 and is due to Held and Karp [151]. They presented an elegant exact algorithm that solves the Traveling Salesman problem in $\mathcal{O}(n^2 2^n)$ time, an impressive improvement over the trivial $\mathcal{O}(n!)$ time complexity. But arguably the main reason why this algorithm became so well-known is the fact that, for almost 50 years now, researchers have not been able to come up with an exact algorithm for the intensively studied Traveling Salesman problem with a better time complexity than the one by Held and Karp.

It is interesting to note that the paper by Held and Karp predates the introduction of the concept of NP-completeness [78] by almost a decade. The lack of a formal framework indicating that the Traveling Salesman problem, like many others, does not admit a polynomial time algorithm did not withhold Held and Karp (or rather the editor of the journal in question) from publishing an exponential time algorithm. After the landmark papers of Cook [78] and Karp [176] provided strong evidence of the intractability of many important combinatorial and graph theoretic problems, the number of publications involving exact exponential time algorithms rapidly increased [172, 189, 209, 234], especially over the past two decades [20, 21, 22, 23, 34, 54, 57, 61, 88, 171, 186]. This surge of interest in exact exponential time algorithms is driven by both practical and theoretical reasons, which will be discussed in Section 1.3.1.

Despite the large number of publications involving exponential time algorithms, no real attempts seemed to have been made to develop a general theory on the subject, until Woeginger [265] wrote a survey paper on exact exponential time algorithms for NP-hard problems in 2003. In this survey, Woeginger describes four of the most widely applicable and successful techniques for designing exact algorithms. He also mentions known results on exact algorithms for several NP-hard problems, and formulates many open problems. Fomin, Grandoni and Kratsch [114] give an overview of some techniques that were not included in Woeginger's survey. Section 1.3.2 briefly discusses the six main techniques that are covered in the survey papers by

Woeginger [264] and Fomin, Grandoni and Kratsch [114], and we refer to those papers for a more detailed introduction. Although we do mention several papers to illustrate the successful application of each of the techniques, we refrain from going into details on how exactly the techniques were used. As before, it is not our intention to give a complete overview of the field, nor do we list all the latest results. Interested readers are advised to consult the aforementioned survey papers, or to get hold of one of the first copies of the forthcoming book "Exact Algorithms" by Fomin and Kratsch [117].

1.3.1 Why exponential time algorithms?

At the beginning of this chapter, we mentioned that an ideal algorithm for solving a problem finds an optimal solution to the problem, finds such a solution for every possible instance, and finds it in polynomial time. In case the problem under consideration requires an exact optimal solution, heuristics and approximation algorithms are not applicable, and improving the running time of the exact algorithm might be the only way to solve larger instances of the problem. And even if approximate solutions were acceptable, some NP-hard problems are hard to approximate, as we already mentioned at the beginning of Chapter 1. Two particularly striking examples of this are the problems Maximum Clique and Chromatic Number. Although both problems have trivial n-approximation algorithms, it is known that neither problem can be approximated in polynomial time within a factor $n^{1-\epsilon}$ for any $\epsilon > 0$, unless P = NP [270]. In other words, finding certain reasonable solutions to some NP-hard problems is just as difficult as computing optimal solutions, which motivates the study of exact exponential time algorithms.

Running an exponential time algorithm on a faster computer increases the size of the instances solvable within a given amount of time by an *additive* constant, whereas a reduction of the base of the exponential running time increases that size by a constant *multiplicative* factor. For example, suppose we are given an algorithm with time complexity 1.7^n and suppose the largest instance we can solve within a "reasonable" amount of time using this algorithm has size n_0. Let n_1 be the size of the largest instance we can solve within the same amount of time, using the same algorithm on a 10-times faster computer. Since $1.7^{n_1} = 10 \cdot 1.7^{n_0}$ yields $n_1 = n_0 + \log_{1.7}(10) \approx n_0 + 4$, we conclude that by using a 10-times faster computer the size of the largest

solvable instance increases by 4. Suppose we manage to improve the algorithm so that it has time complexity 1.3^n, and let n_2 be the size of the largest instance we can solve within the same reasonable amount of time, using this new algorithm on the slow computer mentioned above. Then the maximum size of instances we can handle doubles, since $1.3^{n_2} = 1.7^{n_0}$ yields $n_2 = \log_{1.3}(1.7) \cdot n_0 \approx 2 \cdot n_0$.

Although in theory exponential time algorithms are considered to be inefficient, a fast algorithm with exponential worst-case running time might still be of practical use, and sometimes even outperform a polynomial time algorithm. For example, it is clear that for small instances an algorithm with an exponential time complexity of $\mathcal{O}(1.1^n)$ is expected to run faster than an algorithm with a polynomial time complexity of $\mathcal{O}(n^5)$: note that for example $1.1^{250} \approx 2.2 \cdot 10^{10}$ whereas $250^5 \approx 9.8 \cdot 10^{11}$, which means that for instances up to size 250 the algorithm with the exponential time complexity is preferable over the one with the polynomial time complexity.

The ever-increasing speed of modern computers sometimes allows us to effectively solve large fixed instances of NP-hard problems. In May 2004, the Traveling Salesman problem of visiting all 24,978 cities in Sweden was solved, exceeding the 15,112-city tour through Germany found in 2001 [6]. In 2007, Cook, Espinoza and Goycoolea [79] provided the optimal solution of a 33,810-city instance, and Applegate et al. [7] announced the solution of a Traveling Salesman problem instance of 85,900 cities two years later. There is a big gap between results from testing implementations and known theoretical results on exact algorithms. This gap indicates that there might be a lot of progress to be made and significantly faster exact algorithms to be designed.

Some NP-hard problems have better and faster exact algorithms than others. As an example, let us focus on the Traveling Salesman problem a bit more. We already mentioned at the start of Section 1.3 that the $\mathcal{O}^*(2^n)$ time algorithm for this problem by Held and Karp [151] has not been beaten for almost 50 years. Constructing an exact algorithm with time complexity $\mathcal{O}^*(c^n)$ for some constant $c < 2$ that solves the Traveling Salesman problem is in fact an important open problem, and even finding such an algorithm for the closely related but somewhat easier Hamiltonian Cycle problem would be a very interesting breakthrough [264]. However, for the Euclidean

Traveling Salesman problem, in which the distance between cities is the "ordinary" Euclidean distance, a sub-exponential $\mathcal{O}^*(c^{\sqrt{n}\log n})$ time algorithm for some constant $c > 0$ is known [168], even though this problem is NP-hard on general input [125, 218]. There are many more NP-hard problems that admit sub-exponential time algorithms, i.e., algorithms with time complexity $\mathcal{O}^*(c^{o(n)})$ for some constant $c > 1$ (see [89] for several references). It is natural to ask which NP-hard problems can be solved in sub-exponential (but super-polynomial) time.

The complexity class SNP, a subclass of NP, was introduced by Papadimitriou and Yannakakis [220] for studying the approximability of optimization problems (see [220] for the exact definition). Many important NP-complete problems, such as k-Satisfiability, k-Colorability, Independent Set, and Clique, have been shown to be SNP-complete under so-called SERF-reductions [169]. This implies that if any one of those problems turns out to be solvable in sub-exponential time, then the same holds for every problem in SNP. An equivalent way of stating this is known as the Exponential Time Hypothesis, stating that, for any fixed $k \geq 3$, the k-Satisfiability problem does not have a sub-exponential time algorithm. Since each of the aforementioned problems has withstood numerous attempts of many researchers to find sub-exponential time algorithms [264], the Exponential Time Hypothesis, although perhaps less well-known than its big brother P $\neq$ NP, is widely believed to be true. This is another strong indication that some NP-complete problems might be harder than others. Investigating exact exponential time algorithms for NP-complete problems might lead to a better understanding of the worst-case time behavior of these problems.

1.3.2 Techniques for design and analysis

There is a wide range of techniques that can be used in the design and analysis of good exact algorithms. Some of these techniques appeared in the literature as early as the sixties and seventies [151, 255], others have been developed more recently [113]. Below we give an overview of six important and successful techniques. As mentioned before, we refer to the excellent survey papers by Woeginger [264] and by Fomin, Grandoni and Kratsch [114], as well as to the forthcoming book on the subject by Fomin and Kratsch [117], for more information.

Preprocessing

Preprocessing is a collective term for modifying the input of an algorithm during the initial phase of computation. It normally consists of analyzing, restructuring and/or simplifying the input, but strictly speaking a simple operation like labeling all the vertices of an input graph also counts as preprocessing. A lot of well-known polynomial time sorting algorithms start by preprocessing the input array: for example, the `heapsort` algorithm starts by building a max-heap on the n-element input array, which can be done in linear time, and uses the properties of a heap to efficiently sort the input array in $\mathcal{O}(n \log_2 n)$ time [80]. Obviously, when designing exact exponential time algorithms, we are only interested in preprocessing steps that reduce the time complexity by an *exponential* factor. Sometimes it is not exactly clear what constitutes preprocessing: removing isolated vertices at the start of an algorithm can be seen as preprocessing the input graph, but one might also argue that we are already applying a "proper" (albeit very simple) reduction rule of the algorithm.

Good examples of the successful use of preprocessing in the design of exponential time algorithms are due to Horowitz and Sahny [164]. They consider the Subset Sum problem, asking whether a subset of a given set of integers adds up to a specified value, and the Knapsack problem, where the goal is to find a subset of items with maximum value such that the weight of the items does not exceed a given limit. Both problems can trivially be solved in $\mathcal{O}^*(2^n)$ time by considering all possible subsets. By applying a clever preprocessing trick, involving splitting the input array into two equally sized arrays, Horowitz and Sahny [164] managed to obtain $\mathcal{O}^*(2^{n/2})$ time exact algorithms for both problems. Since the publication of these algorithms in 1974, no faster exact algorithms for the Subset Sum problem and the Knapsack problem have been found.

Dynamic programming

Dynamic programming across subsets is one of the most standard and widely used techniques in the design of exact exponential time algorithms for NP-hard problems. The basic idea is to recursively break the problem down into subproblems, calculate and store the optimal solution to each of the

subproblems, and combine these solutions to find the optimal solution to the original problem. For this method to work, we need the property that optimal solutions of subproblems can be extended to an optimal solution of the original problem.

One of the first and certainly most celebrated results of applying dynamic programming to obtain relatively fast exact algorithms for NP-hard problems is the $\mathcal{O}^*(2^n)$ time algorithm for the Traveling Salesman problem due to Held and Karp [151], already mentioned at the beginning of Section 1.3. Other notable examples include exact algorithms by Lawler [189] for the Chromatic Number problem and the easier 3-Colorability problem, having time complexities of $\mathcal{O}^*(2.443^n)$ and $\mathcal{O}^*(1.443^n)$, respectively. Unlike the algorithm by Held and Karp, the time complexity of which is still unbeaten, faster exact algorithms for Chromatic Number and 3-Colorability are known: these problems can be solved in $\mathcal{O}^*(2^n)$ [34] and $\mathcal{O}^*(1.329^n)$ [23] time, respectively.

Many NP-hard problems can be solved in polynomial, or even linear, time when restricted to the class of trees. At the end of the 1980s, several authors independently discovered that the same holds for many NP-hard problems on graphs whose treewidth is bounded by a constant [13, 26, 37], and a powerful theorem by Courcelle [84] states that *every* decision problem expressible in monadic second order logic can be solved in linear time on graphs of bounded treewidth. Treewidth is a parameter, introduced by Robertson and Seymour [231] in 1986, that indicates how tree-like a graph is (see [40] for an exact definition and more information on treewidth and tree decompositions). Fomin, Grandoni and Kratsch [114] identify dynamic programming over tree decompositions as a powerful technique for designing exact algorithms for graphs whose treewidth is relatively small. Apart from graphs of bounded treewidth, this technique has been successfully applied on planar graphs and on sparse graphs, in particular on graphs with small maximum degree: see the survey paper by Fomin, Grandoni and Kratsch [114] for examples and references.

Pruning the search tree

As we mentioned before, a trivial algorithm for solving an NP-complete problem enumerates and checks all feasible solutions. At any point in the execu-

tion of the algorithm, we can focus on a specific piece of the feasible solution, such as a specific vertex in a graph problem. By determining all possible values for this piece, we can branch into different subcases, each subcase corresponding to a possible value of the piece under consideration. This naturally defines a search tree. If we are lucky, we can determine certain values for this piece that will never lead to an optimal solution. In that case we can "prune" the corresponding subtrees of the search tree, which reduces the search space, and ultimately the overall running time, of the algorithm. An algorithm that uses this technique of *pruning the search tree* is sometimes referred to as a Davis-Putnam-style exponential time backtracking algorithm, and every branch-and-bound algorithm uses this idea.

The technique of pruning the search tree has successfully been used to find good exact algorithms for many NP-hard problems, most notably the 3-Satisfiability problem and the Maximum Independent Set problem. The first exact algorithm for 3-Satisfiability beating a trivial algorithm was obtained by Monien and Speckenmeyer [209] in 1985. Using the technique of pruning the search tree, involving more and more reduction and branching rules, many authors managed to find faster exact algorithms for 3-Satisfiability [**?**, 185, 186, 209, 242, 243] (see [186] for an extensive survey of results of this type). However, the currently fastest exact algorithms for 3-Satisfiability are not based on the technique of pruning the search tree, but use local search ideas, which we will discuss below.

The story for Maximum Independent Set is very similar. After the original breakthrough by Tarjan and Trojanowski [255], who showed that the problem can be solved in $\mathcal{O}^*(2^{n/3}) \approx \mathcal{O}^*(1.260^n)$ time, all the early papers are based on the technique of pruning the search tree [21, 172, 234]. The list of reduction and branching rules grows with the appearance of every subsequent paper, making the case analysis more complex every time. Rather surprisingly, the fastest published polynomial *space* algorithm for Maximum Independent Set is very simple [115]. The crucial idea behind this algorithm, which has a time complexity of $\mathcal{O}^*(1.221^n)$, is the clever choice of a non-standard measure in the analysis of the algorithm. This relatively new idea is called "measure and conquer" and will be discussed below.

Memorization

In a 1986 paper, Robson [234] presents a polynomial space exact algorithm for the Maximum Independent Set problem that runs in $\mathcal{O}^*(1.225^n)$ time. In the same paper, he shows that the time complexity can be reduced to $\mathcal{O}^*(1.211^n)$, although the corresponding algorithm has exponential space complexity. The technique used to obtain this result is called *memorization* by Fomin, Grandoni and Kratsch [114], who include memorization in their survey of new techniques in the design and analysis of exact algorithms. Other authors use the term memoization for the same technique (see for example page 347 of [80] and page 145 of [216]). It turns out that the time complexity of many exponential time search tree algorithms, that typically use a polynomial amount of space, can be reduced at the cost of an exponential space complexity by using this memorization technique [87, ?, 141]. Memorization has also been used in parameterized algorithms [59, 217].

The key idea behind memorization is to store the solution to each subproblem in a database as soon as the recursive algorithm first encounters the subproblem. If a subproblem turns up more than once, the stored solution is looked up to prevent the algorithm from having to compute the same solution again. Since an NP-hard problem with instances of size n typically has an exponential number of subproblems, this database has exponential size. However, if we implement the database in such a way that we can search it in time logarithmic in the number of entries, then we can find a stored subproblem in time polynomial in n.

There are some obvious similarities between memorization and dynamic programming. After all, both techniques involve solving subproblems only once and storing the solutions to these subproblems in a database. However, a bottom-up dynamic programming algorithm will sometimes outperform a top-down memorization-based algorithm, and vice versa. In case all subproblems of a certain problem have to be solved at least once by an algorithm solving the problem, then a dynamic programming algorithm is preferable, since such an algorithm does not involve recursion and needs less time to maintain the table of stored solutions. However, in case we do not need the solution to all subproblems to solve the problem, then a memorization-based algorithm is the better choice. After all, a memorization based algorithm

solves only the subproblems it encounters during execution, whereas a dynamic programming algorithm calculates the solution to *all* the subproblems.

Local search

In the fields of heuristics and randomized algorithms, *local search* has already established itself as a very useful tool in the design of fast algorithms. For example, the technique has played a central role in heuristics for graph coloring problems since the 1980s: Galinier and Hertz [124] give a survey of local search methods used in graph coloring heuristics, classify them into four classes, and conclude that "almost all efficient heuristic algorithms for graph coloring use a local search". Local search starts with a candidate solution (i.e., a random, and therefore not necessarily proper, coloring of a graph) and, in case this candidate is not the required solution to the problem, moves to a "nearby" candidate (for example by recoloring one of adjacent vertices that received the same color in the initial coloring). The meaning of the word "nearby" depends on the context: for many problems, a distance between feasible solutions can be defined naturally. Good local search algorithms use information on why the candidate failed to be a solution to the problem to cleverly choose the next candidate and move a step in the right direction. The choice of the next candidate solution is typically made with a certain degree of randomness. For example, this has led to several local search algorithms for randomized k-Satisfiability [244, 162, 17, 171] (see also the concise survey by Schöning [245] on this topic).

Among the techniques for designing *exact* algorithms, local search is a relative newcomer. In case we are looking for an exact algorithm, we have to make sure that the algorithm visits every possible candidate solution in the search space. Dantsin et al. [88] make clever use of a so-called cover code to derandomize a probabilistic local search algorithm for k-Satisfiability due to Schöning [244], thereby obtaining the then fastest deterministic algorithms for k-Satisfiability for every value of $k \geq 3$. In particular, they show that 3-Satisfiability can be solved in $\mathcal{O}^*(1.481^n)$ time. Brueggemann and Kern [54] slightly improve the pruning technique in [88] and bring the time complexity down to $\mathcal{O}^*(1.473^n)$. This is currently the fastest deterministic exact algorithm for 3-Satisfiability, which shows that local search, despite being a relatively new technique in the area of exact exponential

time algorithms, has already proven to be a very successful one. It is also worth pointing out that the best algorithms for 3-SATISFIABILITY based on local search [88, 54] are much simpler than the ones based on the technique of pruning the search tree [186]: the analysis of the algorithm by Kullmann [186] alone spans 35 pages!

Measure and conquer

Most of the fastest known exact algorithms for NP-hard problems are recursive search tree algorithms. These algorithms use two types of rules: reduction rules to simplify the instance, and branching rules to recursively call the algorithm on smaller instances of the problem. The analysis of these algorithms is based on the bounded search tree technique: after defining a suitable measure of the size of the subproblems, this measure is used to lower bound the progress the algorithm makes at each branching step. This often results in a linear recurrence for every reduction rule and branching rule, which can be solved using standard techniques. Eventually a running time of the type $\mathcal{O}^*(c^n)$ is obtained, for some constant c, by taking the worst case over all the recurrences.

For the last few decades most successful attempts to improve the running time of exact algorithms have been focussed on introducing more and more reduction and branching rules. As a result, many of the currently fastest algorithms are rather complicated. Strangely enough, the measures used in the analysis of these algorithms are usually very simple, for example the number of vertices in case of graph problems or the number of variables or clauses in case of satisfiability problems. Since adding more rules to an already complicated algorithm in order to slightly reduce the running time even further does not seem very inviting, some researchers are investigating the use of non-standard measures in the analysis of algorithms to achieve better upper bounds on the worst-case running times. This relatively new approach is called *measure and conquer*. Note that measure and conquer differs from the techniques discussed in the previous sections in the sense that it is used to *analyze*, rather than design, exact algorithms.

Fomin, Grandoni and Kratsch [114] are the first authors to describe how the use of a non-standard measure can be applied as a general technique to analyze search tree algorithms, and they were the first to use the term

measure and conquer. However, a few years earlier Eppstein already successfully used non-standard measures in the analysis of a search tree algorithms. In [97], he described a simple algorithm for solving the Traveling Salesman problem for cubic graphs, and used a non-standard measure to show that his algorithm runs in $\mathcal{O}^*(1.260^n)$ time. Very recently, Iwama and Nakashima [**?**] improved on this by presenting an $\mathcal{O}^*(1.251^n)$ time algorithm for the same problem. Beigel and Eppstein's [23] $\mathcal{O}^*(1.329^n)$ time exact algorithm for the 3-Coloring problem, that uses a non-standard measure in the analysis, is currently the fastest one known. Other results contributed to the use of the measure and conquer technique include the fastest known exact algorithms for finding a dominating set [235], a minimum connected dominating set [116], and a minimum independent dominating set [128], and many more (see also [117]). These results show that, although the measure and conquer technique was only discovered a few years ago, its results so far have been impressive.

1.4 Thesis overview

This thesis is based on eight research papers [50, 155, 156, 157, 158, 159, 160, 161]. As a result, the problems studied in Chapters 2 to 7 are quite diverse. Therefore, rather than surveying related results on those problems in a separate chapter, we have chosen to start every chapter with a section "Background and results", in which we give an overview of previous work on the problems studied in that chapter, state our contribution in detail, and place our results in a bigger context. For the same reason, we state open problems and suggestions for further research in the last section of each chapter, rather than adding a separate chapter at the end of this thesis.

Graphs without long induced paths play a central role in Chapters 2 and 3. Chapter 2 focuses on the problem of finding certain types of connected dominating subgraphs of such graphs. Our main result is a new characterization of the class of P_6-free graphs in terms of connected dominating subgraphs. In Chapter 3 we study two problems that ask for a partition of the vertex set of the input graph into connected sets. We determine the computational complexity of both problems when restricted to the class of P_k-free graphs, for every k. We also present exact exponential time algorithms for

NP-complete cases of the problems.

Chapter 4 focuses on two problems that ask whether an input graph G contains a smaller graph H as a "pattern". We show that the problem of deciding if G contains a fixed graph H as an induced minor can be solved in polynomial time if H is planar and G belongs to any non-trivial minor-closed graph class. We also present computational complexity results on the problem of deciding whether a given graph G can be contracted to a fixed graph H, as well as on a new variant of this problem. The presence of a dominating vertex in the target graph H seems to play an interesting role in these results.

A k-role assignment of a graph is an assignment of exactly k "roles" to the vertices of the graph, such that two vertices with the same role have exactly the same set of roles in their neighborhood (see Chapter 5 for a more formal definition). The problem of finding a k-role assignment is NP-hard on general graphs when $k \geq 2$. In Chapter 5 we present a polynomial time algorithm for finding a 2-role assignment of a chordal graph. We also show that the problem remains NP-hard when restricted to chordal graphs for any $k \geq 3$.

Chapters 6 and 7 have in common that we restrict the problems studied in both chapters to the class of claw-free graphs. The problems studied in Chapter 6 involve finding induced paths of odd or even length between two specified vertices in a graph. Our main result is a polynomial time algorithm for finding an induced path of given parity between two specified vertices in a claw-free graph. The problem of finding a longest cycle in a graph, the topic of Chapter 7, remains NP-hard on claw-free graphs. We present two exact exponential time algorithms for this problem.

Chapter 2

A new characterization of P_6-free graphs

This chapter is based on the following two papers; the second paper is the extended journal version of the first paper.

[157] P. van 't Hof and D. Paulusma. A new characterization of P_6-free graphs. In: *Proceedings of the 14th Annual International Computing and Combinatorics Conference (COCOON 2008)*, volume 5092 of *Lecture Notes in Computer Science*, pages 415–424, Springer, 2008.

[158] P. van 't Hof and D. Paulusma. A new characterization of P_6-free graphs. *Discrete Applied Mathematics*, 158(7):731–740, 2010.

As the title of this thesis promises, exploiting structural graph properties has played a major role in the design of the algorithms for solving (restricted versions of) the various NP-hard problems studied in this thesis. For example, in Chapter 5 we make use of the well-known clique tree structure of chordal graphs, whereas an interesting structural characterization of claw-free perfect graphs is exploited in Chapter 6. Even more so than in the other chapters of this thesis, the emphasis in this chapter is on structure. We study the class of P_ℓ-free graphs for several values of ℓ. In particular, we focus on characterizations of these graph classes in terms of connected dominating subgraphs. Although the main contribution of this chapter is a new structural characterization of the class of P_6-free graphs, this chapter is also interesting from an algorithmic point of view. The reason for this is that

all our structural results are proved constructively. To be more precise, we present a polynomial time algorithm for finding certain connected dominating subgraphs in P_6-free graphs, as well as polynomial time algorithms for finding such subgraphs in P_4-free and P_5-free graphs. We will use the characterization of P_6-free graphs presented in this chapter to obtain the time complexities of the exact algorithms presented in Chapter 3.

2.1 Background and results

Unless specifically stated otherwise, all graphs in this chapter are non-trivial, i.e., contain at least two vertices. We mentioned in Section 1.2.2 that the class of P_2-free graphs and the class of P_3-free graphs can trivially be characterized as the class of graphs without any edge and the class of graphs all components of which are complete graphs, respectively. We also saw that the class of P_4-free graphs (or cographs) has been studied extensively [47]. Wolk [267, 268] studied the class of *trivially perfect* graphs, which are exactly the graphs that are P_4-free and C_4-free, i.e., the class $Forb(\{P_4, C_4\})$. He showed that a graph G is P_4-free and C_4-free if and only if each connected induced subgraph of G contains a dominating vertex (see also Theorem 11.3.4 in [47]). We show in Section 2.3 that we can slightly generalize this theorem to obtain the following characterization of P_4-free graphs.

Theorem 2.1. *A graph G is P_4-free if and only if each connected induced subgraph of G contains a dominating induced C_4 or a dominating vertex.*

There are many other characterizations of the class of P_4-free graphs in the literature (cf. [47]). We mention one by Bacsó and Tuza [15], obtained independently by Cozzens and Kelleher [85], who show that a graph G is P_4-free if and only if, in every connected induced subgraph G' of G, all maximal cliques dominate G'. Brandstädt et al. [47] included this result in their book as one of many characterizations of cographs (see Theorem 11.3.5 in [47]).

Apart from characterizing the class of P_4-free graphs, Bacsó and Tuza [15] also characterize the class $Forb(\{P_5, C_5\})$. There, they prove that a graph G is P_5-free and C_5-free if and only if each connected induced subgraph of G contains a dominating clique. The same result has been independently obtained by Cozzens and Kelleher [85]. Liu and Zhou [198] improve this by

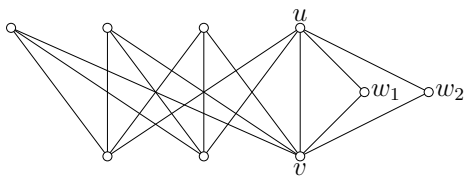

Figure 2.1: An example of a TECB graph.

obtaining the following characterization of P_5-free graphs.

Theorem 2.2 ([198]). *A graph G is P_5-free if and only if each connected induced subgraph of G contains a dominating induced C_5 or a dominating clique.*

A graph G is called *triangle extended complete bipartite (TECB)* if it is a complete bipartite graph or if it can be obtained from a complete bipartite graph F by adding some extra vertices $w_1, \ldots, w_r$ and edges $w_i u, w_i v$ for $1 \leq i \leq r$ to exactly one edge uv of F (see Figure 2.1 for an example).

The following characterization of P_6-free graphs is due to Liu, Peng and Zhao [199].

Theorem 2.3 ([199]). *A graph G is P_6-free if and only if each connected induced subgraph of G contains a dominating induced C_6 or a dominating (not necessarily induced) TECB graph.*

If we consider graphs that are not only P_6-free but also triangle-free, then we have one of the main results in [198].

Theorem 2.4 ([198]). *A triangle-free graph G is P_6-free if and only if each connected induced subgraph of G contains a dominating induced C_6 or a dominating complete bipartite graph.*

A characterization of P_ℓ-free graphs for $\ell \geq 7$ is given in [16]: $Forb(\{P_\ell\})$ is the class of graphs for which each connected induced subgraph has a dominating subgraph of diameter at most $\ell - 4$.

The main result of this chapter, the proof of which is presented in Section 2.4, is the following characterization of the class of P_6-free graphs.

Theorem 2.5. *A graph G is P_6-free if and only if each connected induced subgraph of G contains a dominating induced C_6 or a dominating (not necessarily induced) complete bipartite graph. Moreover, we can find such a dominating subgraph of G in $\mathcal{O}(n^3)$ time.*

This theorem strengthens Theorem 2.3 and Theorem 2.4 in two different ways. Firstly, Theorem 2.5 shows that we may omit the restriction "triangle-free" in Theorem 2.4 and that we may replace the class of TECB graphs by its proper subclass of complete bipartite graphs in Theorem 2.3. Secondly, in contrast to the proofs of Theorem 2.3 and Theorem 2.4, the proof of Theorem 2.5 is constructive: we provide an algorithm for finding a desired dominating subgraph of a P_6-free graph G in $\mathcal{O}(n^3)$ time. Note that we cannot use some brute force approach to obtain such a polynomial time algorithm, since a dominating complete bipartite graph might have arbitrarily large size.

To illustrate the incremental technique used to construct the $\mathcal{O}(n^3)$ time algorithm in the proof of Theorem 2.5, we use the same technique to give constructive proofs of Theorem 2.1 and Theorem 2.2 in Section 2.3. We point out that the proof of Theorem 2.2 in [198] is non-constructive, and to our knowledge there is no constructive proof of Theorem 2.1 in the literature. In particular, we present an $\mathcal{O}(n^3)$ time algorithm that finds a dominating induced C_5 or a dominating clique of a P_5-free graph G. Note that we cannot use a brute force approach to find a dominating clique of a P_5-free graph, as such a clique might have arbitrarily large size. Bacsó and Tuza [15], and independently Cozzens and Kelleher [85], present a polynomial time algorithm that finds a dominating clique of a connected graph without an induced P_5 or C_5. When run on a P_5-free graph G that does contain a C_5, the algorithms in [15] and [85] either find a dominating clique of G or terminate and state that G contains an induced C_5. We point out that the algorithm in our proof of Theorem 2.2 always finds a dominating clique or a *dominating* induced C_5 in a P_5-free graph.

As mentioned before, Section 2.4 contains a constructive proof of Theorem 2.5. We also show that the characterization in Theorem 2.5 is minimal in the sense that there exists an infinite family of P_6-free graphs for which a smallest connected dominating subgraph is a (not induced) complete bipartite graph. We would like to mention that the algorithm used to prove Theorem 2.5 also works for an arbitrary (not necessarily P_6-free) graph G: in that case the algorithm either finds a dominating subgraph as described in Theorem 2.5 or finds an induced P_6 in G. We end Section 2.4 by characterizing the class of graphs for which each connected induced subgraph

has a dominating induced C_6 or a dominating *induced* complete bipartite subgraph, again by giving a constructive proof. This class consists of graphs that, apart from P_6, have exactly one more forbidden induced subgraph (the so-called *net*). This generalizes a result by Bacsó, Michalak and Tuza [14].

As an application of our main result, we consider the Hypergraph 2-Colorability problem in Section 2.5, which is the problem of deciding whether a given hypergraph has a 2-coloring. It is well-known that this problem is NP-complete in general [201]. We prove that for the class of hypergraphs with P_6-free incidence graphs the problem becomes polynomial time solvable. Moreover, we show that for any 2-colorable hypergraph H with a P_6-free incidence graph, we can find a 2-coloring of H in polynomial time.

2.2 An outline of the algorithm

In this section we outline the vertex-incremental approach used in the proofs of Theorem 2.1, Theorem 2.2 and Theorem 2.5. In each of the proofs we describe an algorithm that finds a desired dominating set D of an input graph $G = (V, E)$. The algorithm first establishes a *connected order* $\pi = x_1, \ldots, x_n$ of V, i.e., an order $\pi = x_1, \ldots, x_n$ of the vertices of G such that $G_i := G[\{x_1, \ldots, x_i\}]$ is connected for $i = 1, \ldots, n$. It then processes the vertices of G one by one in a vertex-incremental way, i.e., by adding the next vertex in the order π in every step. Assuming that in an earlier step the algorithm has found a desired dominating subgraph D_{i-1} of G_{i-1}, it adds the next vertex x_i and extends the previously found solution. If the set D_{i-1} dominates G_i, the algorithm sets $D_i := D_{i-1}$ and continues with the next step. Otherwise, it uses the set D_{i-1} plus one or more extra vertices of G_i to find a desired dominating set D_i of G_i. We show that such a transformation can be done in polynomial time by making use of a so-called *minimizer*. Since the algorithm only performs n steps, the total running time stays polynomial. For computational complexity purposes, we represent the graph G by its *adjacency matrix*, i.e., the $n \times n$ matrix $A = (a_{ij})$ with rows and columns indexed by the vertices of V such that $a_{uv} = 1$ if $uv \in E$ and $a_{uv} = 0$ otherwise.

In order to explain the concept of a minimizer we need the following

terminology for a graph $G = (V, E)$. Recall that for any vertex v in G and any subset $D \subseteq V$, we write $N_D(v)$ to denote the subset of neighbors of v that are contained in D, i.e., $N_D(v) := N_G(v) \cap D$. A vertex $v' \in V \setminus D$ is called a *D-private neighbor* of a vertex $v \in D$ if $N_D(v') = \{v\}$. Let u, v be a pair of adjacent vertices in a dominating set D of a graph G such that $\{u, v\}$ dominates D. Note that this means D is connected. We call a dominating set $D' \subseteq D$ of G a *minimizer of D for uv* if $\{u, v\} \subseteq D'$ and each vertex of $D' \setminus \{u, v\}$ has a D'-private neighbor in G. It is important to note that a minimizer D' is connected (because u and v are adjacent vertices in D' and $\{u, v\}$ dominates D') as we will use this property in our algorithms. The following lemma states that we can obtain a minimizer D' from D in polynomial time.

Lemma 2.6. *Let D be a dominating set of a graph G, and let $u, v \in D$ be a pair of adjacent vertices such that $\{u, v\}$ dominates $G[D]$. We can find a minimizer of D for uv in $\mathcal{O}(n^2)$ time.*

Proof. Let $D = \{w_1, \ldots, w_q\}$ with $w_1 = u$ and $w_2 = v$ be a dominating set of a graph $G = (V, E)$. Below we explain how we can obtain a minimizer D' of D for uv in $\mathcal{O}(n^2)$ time. We define $S_i := \emptyset$ for $i = 1, \ldots, q$. For each $x \in V$ we compute in $\mathcal{O}(n)$ time the highest index p such that $x \in N(w_p)$ but $x \notin N(w_1) \cup \cdots \cup N(w_{p-1})$ and set $S_p := S_p \cup \{x\}$. Note that such an index p always exists as $uv \in E$, $\{u, v\}$ dominates D, and D dominates G. We then obtain $\mathcal{S} = \{S_1, \ldots, S_q\}$ in $\mathcal{O}(n^2)$ total time. By construction, the non-empty sets in $\mathcal{S}$ form a partition of V. It is clear that $S_1 \neq \emptyset$ as $v \in S_1$ and $S_2 \neq \emptyset$ as $u \in S_2$. We will not include a vertex w_i with $S_i = \emptyset$ in D', since all its neighbors will be in $S_1 \cup \cdots \cup S_{i-1}$.

We cannot define D' as the set $\{w_k \in D \mid S_k \neq \emptyset\}$, as there might be a vertex w_i with $S_i \neq \emptyset$ whose neighbors in S_i are all adjacent to vertices in $\{w_k \in D \mid S_k \neq \emptyset \text{ and } k \geq i + 1\}$. Hence we define a set $R := \emptyset$ and do as follows for $i = q, q - 1, \ldots, 3$ (so for indices in decreasing order). We set $T_i := \emptyset$. For each $x \in S_i$ we check in $\mathcal{O}(n)$ time if there exists an index $p' \geq i + 1$ such that $x \in N(w_{p'})$ and $T_{p'} \neq \emptyset$. If so, then $R := R \cup \{x\}$. Otherwise, we set $T_i := T_i \cup \{x\}$. After setting $T_1 := S_1$ and $T_2 := S_2$ we then obtain $\mathcal{T} = \{T_1, \ldots, T_q\}$ in $\mathcal{O}(n^2)$ total time.

We define $D' := \{w_i \in D \mid T_i \neq \emptyset\}$. By definition, D' is a minimizer

of D for uv if D' dominates G, $\{u, v\} \subseteq D'$, and each vertex of $D' \setminus \{u, v\}$ has a D'-private neighbor in G. Clearly, $\{u, v\} \in D'$ as $T_1 = S_1 \neq \emptyset$ and $T_2 = S_2 \neq \emptyset$. By construction, each vertex in $T_i \subseteq S_i$ is a neighbor of w_i and the non-empty sets in $\mathcal{T}$, together with R in case $R \neq \emptyset$, form a partition of the set of vertices in $S_1 \cup \cdots \cup S_q = V$. Hence, D' dominates $V \setminus R$. Let $x \in R$. We claim that x is adjacent to a vertex in D'. Suppose $x \in S_h$. Because $x \in R$, we have $h \geq 3$. By our algorithm, $x \in N(w_j)$ for some w_j with $j \geq h + 1$ and $T_j \neq \emptyset$. By definition, $w_j \in D'$. Hence, D' dominates G.

We claim that for every $i \geq 3$ each vertex in T_i with $w_i \in D'$ is a D'-private neighbor of w_i in G. This can be seen as follows. First suppose $x \in T_i$ for some $w_i \in D'$ with $i \geq 3$ is a neighbor of some vertex in $\{w_1, \ldots, w_{i-1}\} \cap D'$. Then $x \notin S_i$, and consequently $x \notin T_i$ as $T_i \subseteq S_i$, a contradiction. Now suppose x is a neighbor of some vertex in $\{w_{i+1}, \ldots, w_q\} \cap D'$. Then x is the neighbor of some vertex w_j with $j \geq i + 1$ and $T_j \neq \emptyset$ by definition of D'. Then, by our algorithm, $x \in R$ instead of $x \in T_i$, which is a contradiction. Hence, all vertices of T_i are D'-private neighbors of w_i in G, for every $i \geq 3$. We conclude that D' is indeed a minimizer of D for uv. □

We point out that a connected dominating set D of a graph G may have several minimizers for the same edge depending on the order in which its vertices are considered. Note that the algorithm described in the proof of Lemma 2.6 also *finds* all the private neighbors for each of the vertices of the minimizer; we will use this in the proof of Theorem 2.5.

Example. Consider the graph G and its connected dominating set D in the left-hand side of Figure 2.2. All D-private neighbors are colored black. Note that u and v are two adjacent vertices in D and that $\{u, v\}$ dominates D. That means we can find a minimizer of D for uv by applying the algorithm described in the proof of Lemma 2.6.

Suppose we choose the order $w_1 = u, w_2 = v, w_3, w_4, w_5, w_6$. We first find non-empty sets S_1, S_2, S_3, S_4, S_5 and we conclude that S_6 is empty. Then we find that T_6 is empty, and that T_5, T_4, T_3 as well as $T_2 = S_2$ and $T_1 = S_1$ are non-empty. The set $D' := \{u, v, w_3, w_4, w_5\}$ is a minimizer of D for uv. The right-hand side of Figure 2.2 shows the graph G and the minimizer D' of D for uv: every vertex in $D' \setminus \{u, v\}$ has a black colored D'-private neighbor. Note that u does not have a D'-private neighbor but v does.

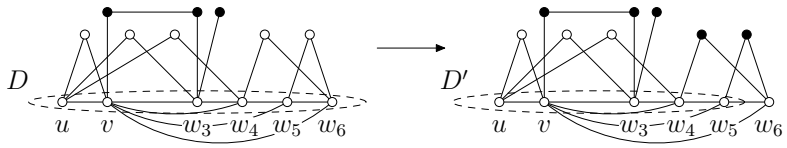

Figure 2.2: A dominating set D and a minimizer D' of D for uv.

If we choose the order $w'_1 = u, w'_2 = v, w'_3 = w_3, w'_4 = w_6, w'_5 = w_4, w'_6 = w_5$ in the algorithm described in the proof of Lemma 2.6, then we find a different minimizer of D for uv. We first find that S'_1, S'_2, S'_3, S'_4 are non-empty and that S'_5, S'_6 are empty. Then we find that T'_6, T'_5 are empty and T'_4, T'_3, T'_2, T'_1 are non-empty. This means that $D'' := \{u, v, w_3, w_6\}$ is a minimizer of D for uv. Note that every vertex of D'' (including u) has a D''-private neighbor.

2.3 Finding connected dominating subgraphs in P_4-free and P_5-free graphs

We now use the technique described in Section 2.2 to prove Theorem 2.1.

Theorem 2.1. *A graph G is P_4-free if and only if each connected induced subgraph of G contains a dominating induced C_4 or a dominating vertex.*

Proof. If G is not P_4-free, then G contains an induced subgraph isomorphic to P_4, and that subgraph has no dominating induced C_4 nor a dominating vertex. So in order to prove Theorem 2.1, it suffices to prove that if G is a connected P_4-free graph, then we can find a dominating induced C_4 or a dominating vertex of G.

Let $G = (V, E)$ be a connected P_4-free graph with connected order $\pi = x_1, \ldots, x_n$. Let $D_1 := \{x_1\}$. Suppose $i \geq 2$. Assume D_{i-1} induces a dominating C_4 in G_{i-1} or is a dominating vertex of G_{i-1}. We write $x := x_i$. If $x \in N(D_{i-1})$, then we set $D_i := D_{i-1}$. Suppose otherwise. We show how we can use D_{i-1} to find a suitable dominating set D_i of G_i, which suffices to prove Theorem 2.1.

Since π is connected, G_i contains a vertex y (not in D_{i-1}) adjacent to x.

Case 1. D_{i-1} induces a dominating C_4 in G_{i-1}.

We write $G[D_{i-1}] = c_1c_2c_3c_4c_1$. Without loss of generality, assume that y is adjacent to c_1. Then y must also be adjacent to c_2 (respectively c_4), as otherwise xyc_1c_2 (respectively xyc_1c_4) is an induced P_4, contradicting the P_4-freeness of G_i. In fact, y must be adjacent to c_3 as well, since otherwise xyc_2c_3 would be an induced P_4 in G_i. If y dominates G_i, then we choose $D_i := \{y\}$. Otherwise, let z be a vertex of G_i not adjacent to y. Since D_{i-1} dominates z, z must be adjacent to at least one vertex c_k of D_{i-1}. The path xyc_kz and P_4-freeness of G_i imply that z must be adjacent to x. Hence the set $C := \{x, y, c_k, z\}$ induces a C_4 in G_i. We claim that C also dominates G_i, which means we can choose $D_i := C$. Suppose C does not dominate G_i, and let z' be a vertex not dominated by C. Since D_{i-1} dominates G_{i-1}, z' must be adjacent to at least one vertex c_ℓ in $D_{i-1} \setminus \{c_k\}$. Then $z'c_\ell yx$ is an induced P_4 in G_i, a contradiction.

Case 2. D_{i-1} **is a dominating vertex of** G_{i-1}**.**

We write $D_{i-1} = \{d\}$. If y dominates G_i, then we choose $D_i := \{y\}$. Otherwise, let z be a vertex of G_i not adjacent to y. Since d dominates z, D_i contains the path $xydz$. The P_4-freeness of G_i implies that z must be adjacent to x. Note that $\{x, y, d, z\}$ dominates G_i, since d dominates every vertex in G_{i-1}. Since $\{x, y, d, z\}$ also induces a C_4 in G_i, we can choose $D_i := \{x, y, d, z\}$. □

Note that we can easily find a dominating vertex or a dominating induced C_4 of a P_4-free graph G in $\mathcal{O}(n^4)$ time by using a brute force approach. Using the technique described in the proof of Theorem 2.1, we can find such a subgraph in $\mathcal{O}(n^2)$ time, as transforming a set D_{i-1} to D_i takes $\mathcal{O}(n)$ time and there are $n - 1$ of such transformations. Note that finding a minimizer was not necessary here.

We now present an algorithmic proof of Theorem 2.2, again using the technique described in Section 2.2.

Theorem 2.2. *A graph G is P_5-free if and only if each connected induced subgraph of G contains a dominating induced C_5 or a dominating clique.*

Proof. If G is not P_5-free, then G contains an induced subgraph isomorphic to P_5, and that subgraph has no dominating induced C_5 nor a dominating clique. So in order to prove Theorem 2.2, it suffices to prove that if G is

a connected P_5-free graph, then we can find a dominating induced C_5 or a dominating clique of G.

Let $G = (V, E)$ be a connected P_5-free graph with connected order $\pi = x_1, \ldots, x_n$. Let $D_1 := \{x_1\}$. Suppose $i \geq 2$. Assume D_{i-1} induces a dominating C_5 in G_{i-1} or is a dominating clique of G_{i-1}. We write $x := x_i$. If $x \in N(D_{i-1})$, then we set $D_i := D_{i-1}$. Suppose otherwise. We show how we can find a suitable dominating set D_i of G_i from D_{i-1}.

Since π is connected, G_i contains a vertex y (not in D_{i-1}) adjacent to x.

Case 1. D_{i-1} induces a dominating C_5 in G_{i-1}.

We write $G[D_{i-1}] = c_1c_2c_3c_4c_5c_1$. Since D_{i-1} dominates G_{i-1} and $y \in V(G_{i-1})$, y must be adjacent to D_{i-1}. Without loss of generality, we assume that y is adjacent to c_1. Obviously, $D^1 := D_{i-1} \cup \{y\}$ dominates G_i. Suppose c_3 has a D^1-private neighbor c_3'. Then $c_3'c_3c_4c_5c_1$ is an induced P_5 in G_i, a contradiction. Hence c_3 has no D^1-private neighbor and $D^2 := D^1 \setminus \{c_3\}$ dominates G_i. Similarly, c_4 has no D^2-private neighbor c_4', since otherwise $c_4'c_4c_5c_1c_2$ would be an induced P_5. So $D^3 := D^2 \setminus \{c_4\} = \{c_1, c_2, c_5, y\}$ still dominates G_i.

Suppose c_2 does not have a D^3-private neighbor. Then $D^4 := \{y, c_1, c_5\}$ dominates G_i. If c_5 has no D^4-private neighbor, then $\{y, c_1\}$ dominates G_i and is a clique of G_i, so we choose $D_i := \{y, c_1\}$. Suppose c_5 has a D^4-private neighbor c_5'. Since $c_5'c_5c_1yx$ is a path on five vertices, we must have $yc_5 \in E(G_i)$ or $xc_5' \in E(G_i)$. If $yc_5 \in E(G_i)$, then $\{y, c_1, c_5\}$ is a clique of G and we choose $D_i := \{y, c_1, c_5\}$. In case $xc_5' \in E(G_i)$, we can choose $D_i := \{x, y, c_1, c_5, c_5'\}$, since $\{x, y, c_1, c_5, c_5'\} \supset D^4$ dominates G_i and induces a C_5 in G.

So we may without loss of generality assume that c_2 has a D^3-private neighbor c_2'. Suppose $yc_2 \notin E(G_i)$. Since $xyc_1c_2c_2'$ is a path on five vertices in G_i, we must have $xc_2' \in E(G_i)$. Let $D := \{x, y, c_1, c_2, c_2'\}$. We claim that D dominates G_i. Suppose, for contradiction, that there exists a vertex $z_1 \notin N_{G_i}(D)$. Since z_1 must be adjacent to D^3, we have $z_1c_5 \in E(G_i)$. But then $z_1c_5c_1c_2c_2'$ is an induced P_5 in G_i, a contradiction. Hence, D dominates G_i. Since $G[D]$ is isomorphic to C_5, we can choose $D_i = D$.

Suppose $yc_2 \in E(G_i)$. If c_5 has no D^3-private neighbor, then $\{y, c_1, c_2\}$ dominates G_i and we choose $D_i := \{y, c_1, c_2\}$. Assume that c_5 has a D^3-

private neighbor c_5''. Using similar arguments as before, we may assume that y is adjacent to c_5. Note that the path $c_2'c_2yc_5c_5''$ cannot be induced in G_i, so c_2' must be adjacent to c_5''. Let $D' := \{c_2', c_2, y, c_5, c_5''\}$. We claim that D' dominates G_i. Suppose D' does not dominate G_i. Then there exists a vertex $z \notin N_{G_i}(D')$. Recall that $D^3 = \{c_1, c_2, c_5, y\}$ dominates G_i. Hence z must be adjacent to c_1. But then $zc_1c_2c_2'c_5''$ induces a P_5 in G_i, a contradiction. Hence D' dominates G_i. Since $G[D']$ is isomorphic to C_5, we can choose $D_i := D'$.

Case 2. D_{i-1} **is a dominating clique of** G_{i-1}**.**

Let y be adjacent to $d_1 \in D_{i-1}$. Since $\{y, d_1\}$ dominates $D_{i-1} \cup \{y\}$, we can compute a minimizer D of $D_{i-1} \cup \{y\}$ for yd_1 by Lemma 2.6. If y is adjacent to all vertices in $D \setminus \{y\}$, then D is a clique and we choose $D_i := D$. Suppose otherwise. Let d_2 be not adjacent to y. By the definition of a minimizer, d_2 has a D-private neighbor d_2'. Since the path $xyd_1d_2d_2'$ cannot be induced in G_i, we have $xd_2' \in E(G_i)$. We claim that the induced cycle $C := xyd_1d_2d_2'x$ dominates G_i. Suppose C does not dominate G_i. Then there exists a vertex $z \notin N_{G_i}(C)$. Since D dominates G_{i-1}, z must be adjacent to a vertex $d \in D \setminus \{d_1, d_2, y\}$. Then $zdd_2d_2'x$ is an induced P_5 in G_i, a contradiction. Hence we can choose $D_i := V(C)$. □

A closer analysis of the proof of Theorem 2.2 shows that Case 1 takes $\mathcal{O}(n)$ time, while Case 2 takes $\mathcal{O}(n^2)$ time if we apply Lemma 2.6 to compute the minimizer. Since there are $n-1$ transformations, we find the following corollary. Note that we cannot use some brute force approach to find such a subgraph, since a dominating clique might have arbitrarily large size.

Corollary 2.7. *We can find a dominating induced C_5 or a dominating clique of a connected P_5-free graph in $\mathcal{O}(n^3)$ time.*

2.4 Finding connected dominating subgraphs in P_6-free graphs

In this section we present a constructive proof of our main result, Theorem 2.5. Let G be a connected P_6-free graph. We say that D is a *type 1 dominating set* of G if D dominates G and $G[D]$ is an induced C_6. We

say that D is a *type 2 dominating set* of G *defined by* $A(D)$ and $B(D)$ if D dominates G and $G[D]$ contains a spanning complete bipartite subgraph with partition classes $A(D)$ and $B(D)$. In the proof of Theorem 2.5 below we present an algorithm that finds a type 1 or type 2 dominating set of G in polynomial time by using the incremental technique described in Section 2.2.

Theorem 2.5. *A graph G is P_6-free if and only if each connected induced subgraph of G contains a dominating induced C_6 or a dominating (not necessarily induced) complete bipartite graph. Moreover, we can find such a dominating subgraph of G in $\mathcal{O}(n^3)$ time.*

Proof. If a graph is not P_6-free, it contains an induced P_6 which contains neither a dominating induced C_6 nor a dominating complete bipartite graph. So to prove Theorem 2.5, it suffices to prove that if G is a connected P_6-free graph, then we can find a type 1 or type 2 dominating set of G in $\mathcal{O}(n^3)$ time.

Let $G = (V, E)$ be a connected P_6-free graph with connected order $\pi = x_1, \ldots, x_n$. Let $D_2 := \{x_1, x_2\}$. Suppose $i \geq 3$. Assume D_{i-1} is a type 1 or type 2 dominating set of G_{i-1}. We write $x := x_i$. If $x \in N(D_{i-1})$, which we can check in $\mathcal{O}(n)$ time, then we set $D_i := D_{i-1}$. Suppose otherwise. We show how we can use D_{i-1} to find D_i in $\mathcal{O}(n^2)$ time. Since the total number of iterations is $n-2$, we then find a desired dominating subgraph of $G_n = G$ in $\mathcal{O}(n^3)$ time.

Since π is connected, G_i contains a vertex y (not in D_{i-1}) adjacent to x. We can find such a vertex in $\mathcal{O}(n)$ time.

Case 1. D_{i-1} is a type 1 dominating set of G_{i-1}.

We write $G[D_{i-1}] = c_1c_2c_3c_4c_5c_6c_1$. We claim that $D := N_{D_{i-1}}(y) \cup \{x, y\}$ dominates G_i, which means that $D_i := D$ is a type 2 dominating set of G_i that is defined by $A(D_i) := \{y\}$ and $B(D_i) := \{x\} \cup N_{D_{i-1}}(y)$ and that can be obtained in $\mathcal{O}(n)$ time. Suppose D does not dominate G_i, and let $z \in V(G_i)$ be a vertex not dominated by D. Since D_{i-1} dominates G_{i-1}, we may without loss of generality assume that $yc_1 \in E(G_i)$.

Suppose $yc_4 \in E(G_i)$. Note that z is dominated by G_{i-1}. Without loss of generality, assume z is adjacent to c_2. Consequently, y is not adjacent to c_2. Since z is not adjacent to any neighbor of y and the path $zc_2c_1yc_4c_5$ cannot

be induced in G_i, either z or y must be adjacent to c_5. If $zc_5 \in E(G_i)$, then $xyc_4c_5zc_2$ is an induced P_6 in G_i. Hence $zc_5 \notin E(G_i)$ and $yc_5 \in E(G_i)$. In case $zc_6 \in E(G_i)$ we obtain an induced path $xyc_5c_6zc_2$ on six vertices, and in case $zc_6 \notin E(G_i)$ we obtain an induced path $zc_2c_1c_6c_5c_4$. We conclude $yc_4 \notin E(G_i)$.

Suppose y is not adjacent to any vertex in $\{c_3, c_5\}$. Since G_i is P_6-free and $xyc_1c_2c_3c_4$ is a P_6 in G_i, y must be adjacent to c_2. But then $xyc_2c_3c_4c_5$ is an induced P_6 in G_i, a contradiction. Hence y is adjacent to at least one vertex in $\{c_3, c_5\}$, say $yc_5 \in E(G_i)$. By symmetry (using c_5, c_2 instead of c_1, c_4) we find $yc_2 \notin E(G_i)$.

Suppose z is adjacent to c_2. The path $zc_2c_1yc_5c_4$ on six vertices and the P_6-freeness of G_i imply $zc_4 \in E(G_i)$. But then $c_2zc_4c_5yx$ is an induced P_6. Hence $zc_2 \notin E(G_i)$. Also $zc_4 \notin E(G_i)$ as otherwise $zc_4c_5yc_1c_2$ would be an induced P_6, and $zc_3 \notin E(G_i)$ as otherwise $zc_3c_2c_1yx$ would be an induced P_6. Then z must be adjacent to c_6 yielding an induced path $zc_6c_1c_2c_3c_4$ on six vertices. Hence we may choose $D_i := D$.

Case 2. D_{i-1} is a type 2 dominating set of G_{i-1}.

Since D_{i-1} dominates G_{i-1}, we may assume that y is adjacent to some vertex $a \in A(D_{i-1})$. Let $b \in B(D_{i-1})$. Note that a and b are adjacent vertices in $D_{i-1} \cup \{y\}$ and that $\{a, b\}$ dominates $D_{i-1} \cup \{y\}$. Hence we can find a minimizer D of $D_{i-1} \cup \{y\}$ for ab in $\mathcal{O}(n^2)$ time by Lemma 2.6. By definition, D dominates G_i. Also, $G[D]$ contains a spanning (not necessarily complete) bipartite graph with partition classes $A \subseteq A(D_{i-1}), B \subseteq B(D_{i-1}) \cup \{y\}$. Note that we have $y \in D$, because x is not adjacent to D_{i-1} and therefore x is a D-private neighbor of y. Since y might not have any neighbors in B but does have a neighbor (vertex a) in A, we chose $y \in B$.

Claim 1. If $G[D]$ contains an induced P_4 starting in y and ending in some $r \in A$, then we can find a type 1 or a type 2 dominating set D_i of G_i in $\mathcal{O}(n^2)$ time.

We prove Claim 1 as follows. Suppose $ypqr$ is an induced path in $G[D]$ with $r \in A$. Since D is a minimizer of $D_{i-1} \cup \{y\}$ for ab and $r \in D \setminus \{a, b\}$, r has a D-private neighbor s by definition. We already identified s when running the algorithm of Lemma 2.6. Since $xypqrs$ is a path on six vertices

and $x \notin N(D_{i-1})$ holds, x must be adjacent to s. We first show that $D^1 := N_D(y) \cup \{x, y, q, r, s\}$, obtained in $\mathcal{O}(n)$ time, dominates G_i.

Suppose D^1 does not dominate G. Then there exists a vertex $z \in N(D) \setminus N(D^1)$. Note that $G[(D \setminus \{y\}) \cup \{z\}]$ is connected because the edge ab makes $D \setminus \{y\}$ connected and $\{a, b\}$ dominates D. Let P be a shortest path in $G[(D \setminus \{y\}) \cup \{z\}]$ from z to a vertex $p_1 \in N_D(y)$ (possibly $p_1 = p$). Since $z \notin N(D^1)$ and $p_1 \in D^1$, we have $|V(P)| \geq 3$. This means that $Pyxs$ is an induced path on at least six vertices, unless $r \in V(P)$, since r is adjacent to s. However, if $r \in V(P)$, then the subpath $z\overrightarrow{P}r$ of P from z to r has at least three vertices, because $z \notin N(D^1)$ and $r \in D_1$. This means that $z\overrightarrow{P}rsxy$ contains an induced P_6, a contradiction. Hence D^1 dominates G_i.

To find a type 1 or type 2 dominating set D_i of G_i, we transform D^1 into D_i in $\mathcal{O}(n^2)$ time as follows. Suppose q has a D^1-private neighbor q'. Then $q'qpyxs$ is an induced P_6 in G_i, a contradiction. Hence q has no D^1-private neighbor and the set $D^2 := D^1 \setminus \{q\}$ still dominates G_i. Similarly, r has no D^2-private neighbor r', since otherwise $r'rsxyp$ would be an induced P_6 in G_i. So the set $D^3 := D^2 \setminus \{r\}$ also dominates G_i. Now suppose s does not have a D^3-private neighbor. We can check this in $\mathcal{O}(n^2)$ time. Then the set $D^3 \setminus \{s\}$ dominates G_i. In that case, we find a type 2 dominating set D_i of G_i defined by $A(D_i) := \{y\}$ and $B(D_i) := N_D(y) \cup \{x\}$. Assume that we found a D^3-private neighbor s' of s in G_i. Let $D^4 := D^3 \cup \{s'\}$.

Suppose $N_D(y) \setminus \{p\}$ contains a vertex p_2 that has a D^4-private neighbor p_2'. Then $p_2'p_2yxss'$ is an induced P_6, contradicting the P_6-freeness of G_i. Hence we can remove all vertices of $N_D(y) \setminus \{p\}$ from D^4, and the resulting set $D^5 := \{p, y, x, s, s'\}$ still dominates G_i. We claim that $D^6 := D^5 \cup \{q\}$ is a type 1 dominating set of G_i. Clearly, D^6 dominates G_i, since $D^5 \subseteq D^6$. Since $qpyxss'$ is a P_6 and $qpyxs$ is induced, q must be adjacent to s'. Hence D^6 is a type 1 dominating set of G_i, and we choose $D_i := D^6$. This proves Claim 1.

Let $A_1 := N_A(y)$ and $A_2 := A \setminus A_1$. Let $B_1 := N_B(y)$ and $B_2 := B \setminus (B_1 \cup \{y\})$. We can obtain these sets in $\mathcal{O}(n)$ time. Since $a \in A_1$, we have $A_1 \neq \emptyset$. If $A_2 = \emptyset$, then we define a type 2 dominating set D_i of G_i by $A(D_i) := A$ and $B(D_i) := B$. Suppose $A_2 \neq \emptyset$. Note $|B| \geq 2$, because $\{b, y\} \subseteq B$. If $B_2 = \emptyset$, then we define D_i by $A(D_i) := A \cup \{y\}$ and $B(D_i) := B_1 = B \setminus \{y\}$. Suppose $B_2 \neq \emptyset$. We check in $\mathcal{O}(n^2)$ time if $G[A_1 \cup A_2]$ contains a spanning

complete bipartite graph with partition classes A_1 and A_2. If so, we define D_i by $A(D_i) := A_1$ and $B(D_i) := A_2 \cup B$. Otherwise we have found two non-adjacent vertices $a_1 \in A_1$ and $a_2 \in A_2$. Let $b^* \in B_2$. Then $ya_1b^*a_2$ is an induced P_4 starting in y and ending in a vertex of A. By Claim 1, we can find a type 1 or type 2 dominating set D_i of G_i in $\mathcal{O}(n^2)$ extra time. This finishes the proof of Theorem 2.5. □

The characterization in Theorem 2.5 is minimal in some sense due to the existence of the following family $\mathcal{F}$ of P_6-free graphs. For each $i \geq 2$, let $F_i \in \mathcal{F}$ be the graph obtained from a complete bipartite subgraph with partition classes $X_i = \{x_1, \ldots, x_i\}$ and $Y_i = \{y_1, \ldots, y_i\}$ by adding the edge x_1x_2 as well as for each $h = 1, \ldots, i$ a new vertex x'_h adjacent only to x_h and a new vertex y'_h adjacent only to y_h (see Figure 2.3 for the graph F_3).

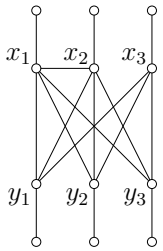

Figure 2.3: The graph F_3.

Note that each F_i is P_6-free and that the smallest connected dominating subgraph of F_i is $F_i[X_i \cup Y_i]$, which contains a spanning complete bipartite subgraph. Also note that none of the graphs F_i contain a dominating *induced* complete bipartite subgraph due to the edge x_1x_2.

We conclude this section by characterizing the class of graphs for which each connected induced subgraph contains a dominating induced C_6 or a dominating *induced* complete bipartite subgraph. Again, we will show how to find these dominating induced subgraphs in polynomial time by using the incremental technique outlined in Section 2.2. The *net* is the graph on six vertices depicted in Figure 2.4.

Theorem 2.8. *A graph G is in $Forb(\{P_6, net\})$ if and only if each connected induced subgraph of G contains a dominating induced C_6 or a dominating induced complete bipartite graph. Moreover, we can find such a dominating subgraph of G in $\mathcal{O}(n^4)$ time.*

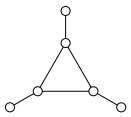

Figure 2.4: The net.

Proof. Neither the graph P_6 nor the net has a dominating induced C_6 or a dominating induced complete bipartite subgraph. Hence to prove Theorem 2.8, it suffices to show that if G is a connected graph in $Forb(\{P_6, net\})$, then we can find a dominating induced C_6 or a dominating induced complete bipartite subgraph of G in $\mathcal{O}(n^4)$ time.

Let $G = (V, E)$ be a connected graph in $Forb(\{P_6, net\})$ with connected order $\pi = x_1, \ldots, x_n$. Recall that we write $G_i := G[\{x_1, \ldots, x_i\}]$, and note that $G_i \in Forb(\{P_6, net\})$ for every i. For every $2 \leq i \leq n$ we want to find a dominating set D_i of G_i that either induces a C_6 or a complete bipartite subgraph in G_i. Let $D_2 := \{x_1, x_2\}$. Suppose $i \geq 3$. Assume D_{i-1} induces a dominating C_6 or a dominating complete bipartite subgraph in G_{i-1}. We show how we can use D_{i-1} to find D_i in $\mathcal{O}(n^3)$ time. Since the total number of iterations is $n-2$, we find a desired dominating subgraph of $G_n = G$ in $\mathcal{O}(n^4)$ time. We write $x := x_i$ and check in $\mathcal{O}(n)$ time if $x \in N(D_{i-1})$. If so then we set $D_i := D_{i-1}$. Suppose otherwise. Since π is connected, G_i contains a vertex y (not in D_{i-1}) adjacent to x. We can find y in $\mathcal{O}(n)$ time. We first prove a useful claim.

Claim 1. If $N_{D_{i-1}}(y) \cup \{x, y\}$ dominates G_i, then we can find a dominating induced C_6 or a dominating induced complete bipartite subgraph of G in $\mathcal{O}(n^3)$ time.

We prove Claim 1 as follows. Suppose $D^* := N_{D_{i-1}}(y) \cup \{x, y\}$ dominates G_i. We check whether $G[D^*]$ is complete bipartite in $\mathcal{O}(n^2)$ time. If so, then we choose $D_i := D^*$ and we are done. Otherwise y has a neighbor u in D_{i-1} with $N_{D^*}(u) \setminus \{y\} \neq \emptyset$. If u has no D^*-private neighbor, which we can check in $O(n^2)$ time, then we remove u from D^* and perform the same check in the smaller set $D^* \setminus \{u\}$. Let u' be a D^*-private neighbor of u in G_i. Let $v \in N_{D^*}(u) \setminus \{y\}$. Then u' is adjacent to any D^*-private neighbor v' of v, as otherwise $G[\{u, v, y, u', v', x\}]$ is isomorphic to the net. So we find that $D^1 := (D^* \setminus N_{D^*}(u)) \cup \{y, u'\}$ dominates G_i. If u' does

not have a D^1-private neighbor, then we remove u' from D^1, check if y is adjacent to two neighbors in the smaller set $D^1 \setminus \{u'\}$ and repeat the above procedure which runs in $\mathcal{O}(n^3)$ total time. Let u'' be a D^1-private neighbor of u'. Suppose $N_{D^1}(y) = \{x, u\}$. Then $D^1 = \{x, y, u, u'\}$. Recall that y dominates D^*, and therefore $D^1 \setminus \{u'\}$, by definition. If x does not have a D^1-private neighbor, then we choose $D_i := \{y, u, u'\}$. If x has a D^1-private neighbor x', then the P_6-freeness of G_i implies that x' is adjacent to u'', and we choose $D_i := \{x', x, y, u, u', u''\}$.

Suppose $N_{D^1}(y) \setminus \{x, u\} \neq \emptyset$, say y is adjacent to some vertex $t \in D^1 \setminus \{x, u\}$. If t does not have a D^1-private neighbor, then we remove t from D^1 and check if y is adjacent to some vertex in the smaller set $D^1 \setminus \{x, u, t\}$. Let t' be a D^1-private neighbor of t. Note that D^1 does not contain any neighbors of u. Hence the path $u''u'uytt'$ is an induced P_6 of G_i, unless u'' is adjacent to t'. However, in that case $xyuu'u''t'$ is an induced P_6. This contradiction finishes the proof of Claim 1.

Case 1. D_{i-1} induces a dominating C_6 in G_{i-1}.

Since D_{i-1} is a type 1 dominating set of G_{i-1}, we know from the corresponding Case 1 in the proof of Theorem 2.5 that $D := N_{D_{i-1}}(y) \cup \{x, y\}$ dominates G_i. We can find D in $\mathcal{O}(n)$ time. By Claim 1, we can find a dominating induced C_6 or a dominating induced complete bipartite subgraph of G in $\mathcal{O}(n^3)$ extra time.

Case 2. D_{i-1} induces a dominating complete bipartite subgraph in G_{i-1}.

Let $A(D_{i-1})$ and $B(D_{i-1})$ denote the partition classes of D_{i-1}. Note that both $A(D_{i-1})$ and $B(D_{i-1})$ are independent sets. Since D_{i-1} dominates G_{i-1}, we may without loss of generality assume that y is adjacent to some vertex $a \in A(D_{i-1})$. Let $b \in B(D_{i-1})$. Note that a and b are adjacent vertices in $D_{i-1} \cup \{y\}$ and that $\{a, b\}$ dominates $D_{i-1} \cup \{y\}$. Hence we can find a minimizer D of $D_{i-1} \cup \{y\}$ for ab in $\mathcal{O}(n^2)$ time by Lemma 2.6. By definition, D dominates G_i. Also, $G[D]$ contains a spanning (not necessarily complete) bipartite graph with partition classes $A \subseteq A(D_{i-1})$ and $B \subseteq B(D_{i-1}) \cup \{y\}$. Note that $y \in D$, because x is not adjacent to D_{i-1} and therefore x is a D-private neighbor of y, and consequently, $y \in B$ because y is adjacent to $a \in A$ and y might not have any neighbors in B. Let $A_1 := N_A(y)$ and

$A_2 := A \setminus A_1$. Let $B_1 := N_B(y)$ and $B_2 := B \setminus (B_1 \cup \{y\})$. We can obtain these sets in $\mathcal{O}(n)$ time. Since $a \in A_1$, we have $A_1 \neq \emptyset$.

Suppose $G[D]$ contains an induced P_4 starting in y and ending in a vertex in A. Just as in the proof of Theorem 2.5 we can obtain in $\mathcal{O}(n^2)$ time a dominating C_6 of G_i or else we find that $N_D(y) \cup \{x, y\}$, and consequently $N_{D_{i-1}}(y) \cup \{x, y\}$ dominates G_i. In the first case, we choose D_i to be the obtained dominating induced C_6. In the second case, we can find a dominating induced C_6 or a dominating induced complete bipartite subgraph of G in $\mathcal{O}(n^3)$ extra time by Claim 1. So we may assume that $G[D]$ does not contain such an induced P_4. This means that at least one of the sets A_2, B_2 is empty, as otherwise we find an induced path yab_2a_2 for any $a_2 \in A_2$ and $b_2 \in B_2$. We may without loss of generality assume that $A_2 = \emptyset$. Otherwise, in case $B_2 = \emptyset$, we obtain $B = B_1$, which means that y is adjacent to b, so we can reverse the role of A and B. If $B_2 = \emptyset$, then we find that $A_1 \cup B_1 \cup \{y\} \subset N_{D_{i-1}}(y) \cup \{x, y\}$ dominates G_i, and we are done in $\mathcal{O}(n^3)$ extra time as a result of Claim 1. So $B_2 \neq \emptyset$. Let $b_2 \in B_2$.

We claim that $D^2 := A_1 \cup B_2 \cup \{x, y\}$ dominates G_i. Suppose otherwise. Then there exists a vertex b_1' adjacent to some vertex $b_1 \in B_1$ but not adjacent to D^2. Then $G[\{y, a, b_1, x, b_2, b_1'\}]$ is isomorphic to the net, a contradiction. Hence D^2 dominates G_i. From D^2 we construct D_i in $\mathcal{O}(n^2)$ extra time as follows. If x does not have a D^2-private neighbor, then we can choose $D_i := D^2 \setminus \{x\}$, since $G[D^2 \setminus \{x\}]$ is a complete bipartite graph with partition classes A_1 and $B_2 \cup \{y\}$. Suppose x has a D^2-private neighbor x'. If b_2 does not have a D^2-private neighbor, then we remove b_2 from D^2, and check whether B_2 contains another vertex. If not, then we can choose $D_i := A_1 \cup \{x, y\}$, since $G[A_1 \cup \{x, y\}]$ is a complete bipartite graph with partition classes $A_1 \cup \{x\}$ and $\{y\}$. Suppose b_2 has a D^2-private neighbor b_2'. Then the path $x'xyab_2b_2'$ is a path on six vertices, so we must have $x'b_2' \in E$.

We claim that $D^3 := \{x', x, y, a, b_2, b_2'\}$ dominates G_i. Suppose otherwise. Then there exists a vertex c' adjacent to some vertex c in $A_1 \cup B_2$ but not adjacent to a vertex in D^3. Suppose $c \in A_1$. Then $c'cb_2b_2'x'x$ is an induced P_6. Suppose $c \in B_2$. Then $c'cayxx'$ is an induced P_6. So D^3 dominates G_i. Since D^3 also induces a C_6 in G_i, we may choose $D_i := D^3$. This finishes the proof of Theorem 2.8. □

Bacsó, Michalak and Tuza [14] prove (non-constructively) that a graph G is in $Forb(\{C_6, P_6, net\})$ if and only if each connected induced subgraph of G contains a dominating induced complete bipartite graph. Note that Theorem 2.8 immediately implies this result.

2.5 An application of our characterization

The Hypergraph 2-Colorability problem asks whether a given hypergraph has a 2-coloring. This problem, also known as Set Splitting, is NP-complete, even when restricted to hypergraphs for which every hyperedge has size at most 3 [201]. Note that the problem can be solved in polynomial time when restricted to hypergraphs for which every hyperedge has size 2, since that problem is equivalent to the 2-Colorability problem for graphs, i.e., to checking whether a given graph is bipartite. We now present another class of hypergraphs for which the Hypergraph 2-Colorability problem is solvable in polynomial time. We use the characterization of P_6-free graphs in Theorem 2.5 to obtain this result. Let $\mathcal{H}_6$ denote the class of hypergraphs with P_6-free incidence graphs.

Theorem 2.9. *The* Hypergraph 2-Colorability *problem restricted to* $\mathcal{H}_6$ *can be solved in polynomial time. Moreover, for any 2-colorable hypergraph* $H = (Q, \mathcal{S}) \in \mathcal{H}_6$ *we can find a 2-coloring of* H *in* $\mathcal{O}((|Q| + |\mathcal{S}|)^3)$ *time.*

Proof. Let $H = (Q, \mathcal{S}) \in \mathcal{H}_6$ with $|Q| = q$ and $|\mathcal{S}| = s$, and let I be the P_6-free incidence graph of H. We assume that I is connected, as otherwise we just proceed component-wise.

Claim 1. We may without loss of generality assume that $\mathcal{S}$ *does not contain two sets* S_i, S_j *with* $S_i \subseteq S_j$.

We prove Claim 1 as follows. Suppose $S_i, S_j \in \mathcal{S}$ with $S_i \subseteq S_j$. We show that H is 2-colorable if and only if $H - S_j$ is 2-colorable. Clearly, if H is 2-colorable then $H - S_j$ is 2-colorable. Suppose $H - S_j$ is 2-colorable. Let (Q_1, Q_2) be a 2-coloring of $H - S_j$. By definition, $S_i \cap Q_1 \neq \emptyset$ and $S_i \cap Q_2 \neq \emptyset$. Since $S_i \subseteq S_j$, we also have $S_j \cap Q_1 \neq \emptyset$ and $S_j \cap Q_2 \neq \emptyset$, so (Q_1, Q_2) is a 2-coloring of H. This proves Claim 1.

Note that we can reach the situation mentioned in Claim 1 in $\mathcal{O}(q^2s^2)$ time. By Theorem 2.5, we can find a type 1 or type 2 dominating set D of I in $\mathcal{O}((q+s)^3)$ time. Below we show how we use such a dominating set to find a 2-coloring of H in $\mathcal{O}(q+s)$ extra time, assuming H has a 2-coloring. Since I is bipartite, $I[D]$ is bipartite. Let A and B be the partition classes of $I[D]$. Since I is connected, we may without loss of generality assume $A \subseteq Q$ and $B \subseteq \mathcal{S}$. Let $A' := Q \setminus A$ and $B' := \mathcal{S} \setminus B$. We distinguish two cases.

Case 1. *D is a type 1 dominating set of I.*

We write $I[D] = q_1S_1q_2S_2q_3S_3q_1$, so $A = \{q_1, q_2, q_3\}$ and $B = \{S_1, S_2, S_3\}$. Suppose $A' = \emptyset$, so $Q = \{q_1, q_2, q_3\}$. Obviously, H has no 2-coloring. Suppose $A' \neq \emptyset$ and let $q' \in A'$. Since D dominates I, q' has a neighbor, say S_1, in B. If S_2 and S_3 both have no neighbors in A', then $q'S_1q_2S_2q_3S_3$ is an induced P_6 in I, a contradiction. Hence at least one of them, say S_2, has a neighbor in A'.

We claim that the partition (Q_1, Q_2) of Q with $Q_1 := A' \cup \{q_1\}$ and $Q_2 := \{q_2, q_3\}$ is a 2-coloring of H. We have to check that every $S \in \mathcal{S}$ has a neighbor in both Q_1 and Q_2. Recall that S_1 has neighbors q_1 and q_2 and S_3 has neighbors q_1 and q_3. Hence S_1 has a neighbor in both Q_1 and Q_2, and the same holds for S_3. Since S_2 is adjacent to q_2 and has a neighbor in A', S_2 also has a neighbor in both Q_1 and Q_2. It remains to check the vertices in B'. Let $S \in B'$. Since D dominates I and I is bipartite, S has at least one neighbor in A. Suppose S has exactly one neighbor, say q_1, in A. Then $Sq_1S_1q_2S_2q_3$ is an induced P_6 in I, a contradiction. Hence S has at least two neighbors in A. The only problem occurs if S is adjacent to q_2 and q_3 but not to q_1. However, since S_2 is adjacent to q_2 and q_3, S must have a neighbor in A' due to Claim 1. Hence (Q_1, Q_2) is a 2-coloring of H.

Case 2. *D is a type 2 dominating set of I.*

Suppose $A' = \emptyset$. Then $|B| = 1$ as a result of Claim 1. Let $B = \{S\}$ and $q \in A$. Since S is adjacent to all vertices in A, we find that $B' = \emptyset$ as a result of Claim 1. Hence H has no 2-coloring if $|A| = 1$, and H has a 2-coloring $(\{q\}, A \setminus \{q\})$ if $|A| \geq 2$. Suppose $A' \neq \emptyset$. We claim that (A, A') is a 2-coloring of H. This can be seen as follows. By definition, each vertex in $\mathcal{S}$ is adjacent to a vertex in A. Suppose $|B| = 1$ and let $B = \{S\}$. Since S dominates Q and $A' \neq \emptyset$, S has at least one neighbor in A'. Suppose

$|B| \geq 2$. Since every vertex in B is adjacent to all vertices in A, every vertex in $\mathcal{S}$ must have a neighbor in A' as a result of Claim 1. □

2.6 Conclusion

The key contributions of this chapter are the following. We presented a new characterization of the class of P_6-free graphs, which strengthens results of Liu and Zhou [198] and Liu, Peng and Zhao [199]. We used an algorithmic technique to prove this characterization. Our main algorithm efficiently finds for any given connected P_6-free graph a dominating subgraph that is either an induced C_6 or a (not necessarily induced) complete bipartite graph. Besides these main results, we also showed that our characterization is "minimal" in the sense that there exists an infinite family of P_6-free graphs for which a smallest connected dominating subgraph is a (not induced) complete bipartite graph. We also characterized the class $Forb(\{P_6, net\})$ in terms of connected dominating subgraphs, thereby generalizing a result of Bacsó, Michalak and Tuza [14].

Our main algorithm can be useful to determine the computational complexity of decision problems restricted to the class of P_6-free graphs. To illustrate this, we applied this algorithm to prove that the Hypergraph 2-Colorability problem is polynomially solvable for the class of hypergraphs with P_6-free incidence graphs. Are there any other decision problems for which the algorithm is useful? In recent years, several authors studied the classical k-Colorability problem for the class of P_ℓ-free graphs for various combinations of k and ℓ (see Section 1.2.2). Randerath and Schiermeyer [226] provide a polynomial time algorithm for solving the 3-Colorability problem on P_6-free graphs. Their algorithm relies on the Strong Perfect Graph Theorem [64] and is rather complicated. Very recently, Broersma et al. [49] used the characterization of P_6-free graphs presented in this chapter to obtain a simpler algorithm, independent of the Strong Perfect Graph Theorem, for deciding whether a P_6-free graph has a 3-coloring. Hoàng et al. [152] show that for every fixed $k \geq 3$ the k-Colorability problem can be solved in polynomial time for the class of P_5-free graphs. They pose the question whether there exists a polynomial time algorithm to determine if a P_6-free graph can be 4-colored. We leave as an open question whether our char-

acterization of P_6-free graphs in Theorem 2.5 can be used to answer this question.

A natural problem for a given graph class deals with its recognition. We are not aware of any recognition algorithm for P_6-free graphs other than the trivial algorithm that checks for every 6-tuple of vertices whether they induce a path. This might be another interesting direction for future research, considering the following results on recognition of subclasses of P_6-free graphs. Giakoumakis and Vanherpe [136] show that bipartite P_6-free graphs can be recognized in linear time. They do this by extending the techniques developed in [83] for linear time recognition of P_4-free graphs (also see [143]) and by using a decomposition scheme for bipartite graphs from [120]. Brandstädt, Klembt and Mahfud [46] show that triangle-free P_6-free graphs have bounded clique-width. The recognition algorithm they obtain from this result runs in quadratic time. Since the class of P_6-free graphs has unbounded clique-width (cf. [44]), their technique cannot be applied to find a quadratic recognition algorithm for the class of P_6-free graphs.

The next class to consider is the class of P_7-free graphs. Recall that a graph G is P_7-free if and only if each connected induced subgraph of G contains a dominating subgraph of diameter at most three [16]. Using an approach similar to the one described in this chapter, it is possible to find such a dominating subgraph in polynomial time. However, a more important question is whether this characterization of P_7-free graphs can be narrowed down. Also determining the computational complexity of the Hypergraph 2-Colorability problem for the class of hypergraphs with P_7-free incidence graphs is still an open problem. We are not aware of any recognition algorithm for P_7-free graphs, or even for the subclasses of bipartite or triangle-free P_7-free graphs, that outperforms the trivial recognition algorithm that checks for every 7-tuple of vertices whether they induce a path.

Chapter 3

Partitioning graphs into connected parts

This chapter is based on the following two papers; the second paper is the extended journal version of the first paper.

[160] P. van 't Hof, D. Paulusma, and G.J. Woeginger. Partitioning graphs into connected parts. In: *Proceedings of the 4th International Computer Science Symposium in Russia (CSR 2009)*, volume 5675 of *Lecture Notes in Computer Science*, pages 143–154, Springer, 2009.

[161] P. van 't Hof, D. Paulusma, and G.J. Woeginger. Partitioning graphs into connected parts. *Theoretical Computer Science*, 410:4834–4843, 2009.

There are several natural and elementary algorithmic problems that check if the structure of a graph H shows up as a pattern within the structure of another graph G. One of the most well-known problems of this type is the H-Minor Containment problem that asks whether a given input graph G contains a fixed graph H as a minor. A celebrated result by Robertson and Seymour [232] states that the H-Minor Containment problem can be solved in polynomial time for every graph H. For two related problems, the H-Induced Minor Containment problem and the H-Contractibility problem, asking whether an input graph G contains a fixed graph H as an induced minor or a contraction, respectively, the computational complexity picture is not so clear. These two problems are studied in Chapter 4.

The two problems studied in this chapter are related to the H-Contractibility problem. For one of these problems, the Longest Path Contractibility problem, which asks what the longest path is to which a graph can be contracted, this connection is clear. To see how the other problem, the 2-Disjoint Connected Subgraphs problem, is related to H-Contractibility, let us introduce this problem using the concept of a witness structure. An H-witness structure of a graph, defined in Section 1.1.3, can be seen as a partition of its vertex set into $|V(H)|$ connected sets, where the adjacencies between the sets depend on the target graph H. In particular, a P_2-witness structure of a graph G is a partition of $V(G)$ into two adjacent connected sets V_1, V_2. It is trivial to decide whether such sets V_1 and V_2 exist, as this is equivalent to checking whether G is connected and contains at least one edge. However, the problem seems to become a lot harder if V_1 and V_2 are required to contain prescribed sets of vertices, as we will show in Section 3.2.1. This problem is exactly the 2-Disjoint Connected Subgraphs problem. As will become clear in Section 3.1 this problem also has a strong connection with the H-Minor Containment problem. We point out that the characterization of P_6-free graphs obtained in Chapter 2 will make an appearance in this chapter.

3.1 Background and results

Theoretical motivation for research on edge contractions can be found in [52, 103, 193, 194] and comes from hamiltonian graph theory [154] and graph minor theory [232]. Practical applications include surface simplification in computer graphics [3, 60] and cluster analysis of large data sets [77, 148, 179]. In the first practical application, graphical objects are represented using (triangulated) graphs and these graphs need to be simplified. One of the techniques to do this is by using edge contractions. In the second application, graphs are coarsened by means of edge contractions.

As we mentioned at the beginning of this chapter, Robertson and Seymour [232] proved that the H-Minor Containment problem can be solved in polynomial time for every fixed pattern graph H. They obtained this result by designing an algorithm that solves the following problem in polynomial time for any fixed k. It follows from a result by Karp [177] that the problem

is NP-complete when k is a variable part of the input (see also [232]).

Disjoint Connected Subgraphs
Instance: A graph G and mutually disjoint non-empty sets $Z_1, \ldots, Z_t \subseteq V(G)$ such that $\sum_{i=1}^{t} |Z_i| \leq k$.
Question: Do there exist mutually vertex-disjoint connected subgraphs $G_1, \ldots, G_t$ of G such that $Z_i \subseteq V(G_i)$ for $1 \leq i \leq t$?

The first problem studied in this chapter is the 2-Disjoint Connected Subgraphs problem, which is a restriction of the above problem to $t = 2$. We show in Section 3.2.1 that the 2-Disjoint Connected Subgraphs problem is NP-complete even if one of the given sets of vertices has cardinality 2. Since the appearance of the paper on which this chapter is based [161], the 2-Disjoint Connected Subgraphs problem has attracted a considerable amount of interest [142, 173, 223]. We mention here a result obtained by Gray et al. [142], who prove that the 2-Disjoint Connected Subgraphs problem is NP-complete for the class of planar graphs.

The second problem studied in this chapter is inspired by a concept introduced by Blum [36]: the *cyclicity* $\eta(G)$ of a connected graph G is the largest integer ℓ for which G is contractible to the cycle C_ℓ on ℓ vertices. We introduce a similar concept: the *path contractibility number* $\vartheta(G)$ of a graph G is the largest integer ℓ for which G is P_ℓ-contractible. For convenience, we define $\vartheta(G) = 0$ if and only if G is disconnected. The second problem studied in this chapter is the Longest Path Contractibility problem, which asks for the path contractibility number of a given graph G.

Both the 2-Disjoint Connected Subgraphs problem and the Longest Path Contractibility problem deal with partitioning the vertex set of a given graph into connected sets. Since connectivity is a "global" property, both problems are examples of "non-local" problems, which are typically hard to solve exactly (see e.g. [116]). Arguably the most well-known non-local problem is the Traveling Salesman problem, for which no exact algorithm with better time complexity than $\mathcal{O}^*(2^n)$ is known. Another example of a non-local problem is the Connected Dominating Set problem. The fastest known exact algorithm for the Connected Dominating Set problem runs in $\mathcal{O}^*(1.9407^n)$ time [116], whereas for the general (unconnected) version of the Dominating Set problem an $\mathcal{O}^*(1.5063^n)$ time exact algorithm is known [235]. In an attempt to design fast exact algo-

rithms for non-local problems, one can focus on restrictions of the problem to certain graph classes. One family of graph classes of particular interest is the family of graphs that do not contain long induced paths. Several authors have studied restrictions of well-known NP-hard problems, such as the k-Colorability problem and the Maximum Independent Set problem, to the class of P_ℓ-free graphs for several values of ℓ (see Section 1.2.2 for an extensive list of references).

We start in Section 3.2.1 by proving that the 2-Disjoint Connected Subgraphs problem is NP-complete, even if one of the two prescribed sets of vertices has size 2. We then give a complexity classification for the 2-Disjoint Connected Subgraphs problem and the Longest Path Contractibility problem on P_ℓ-free graphs, for each value of ℓ, in Sections 3.2.2 and 3.3.1, respectively. We show that the 2-Disjoint Connected Subgraphs problem restricted to the class of P_ℓ-free graphs is polynomial time solvable if $\ell \leq 4$ and NP-complete otherwise. Then we show that the Longest Path Contractibility problem restricted to the class of P_ℓ-free graphs is polynomial time solvable if $\ell \leq 5$ and NP-complete otherwise. In Sections 3.2.3 and 3.3.2 we present exact algorithms for NP-hard cases of the 2-Disjoint Connected Subgraphs problem and the Longest Path Contractibility problem, respectively. A trivial algorithm solves the Two Disjoint Connected Subgraphs problem in $\mathcal{O}^*(2^n)$ time. Let $\mathcal{G}^{k,r}$ denote the class of graphs all connected induced subgraphs of which have a connected r-dominating set of size at most k (see Section 3.2.3 for an exact definition). We present an exact algorithm, called `SPLIT`, that solves the 2-Disjoint Connected Subgraphs problem for n-vertex graphs in the class $\mathcal{G}^{k,r}$ in $\mathcal{O}^*((f(r))^n)$ time for any fixed k and $r \geq 2$, where

$$f(r) = \min_{0<c\leq 0.5} \left\{ \max \left\{ \frac{1}{c^c(1-c)^{1-c}}, 2^{1-\frac{2c}{r-1}} \right\} \right\}.$$

In particular, `SPLIT` solves the 2-Disjoint Connected Subgraphs problem for any n-vertex P_6-free graph in $\mathcal{O}^*(1.5790^n)$ time, and beats the trivial $\mathcal{O}^*(2^n)$ time algorithm for solving the problem on P_ℓ-free graphs for every fixed ℓ. We also use `SPLIT` as a subroutine in an $\mathcal{O}^*(1.5790^n)$ time exact algorithm for the Longest Path Contractibility problem restricted to P_6-free graphs.

3.2 The 2-Disjoint Connected Subgraphs problem

We mentioned at the beginning of this chapter that the problem of deciding whether the vertices of a graph can be partitioned into two disjoint connected subgraphs can trivially be solved by testing whether the graph is connected and has at least one edge. In Section 3.2.1 we show that the problem becomes NP-complete if we demand the two disjoint connected subgraphs to contain certain prescribed sets of vertices. In fact, we show the problem is already NP-complete if one of those prescribed sets consists of just two vertices. In Section 3.2.2 we provide a complexity classification of the 2-Disjoint Connected Subgraphs problem restricted to P_ℓ-free graphs for every ℓ. For the classes of P_ℓ-free graphs for which the problem turns out to be NP-complete, we present an exact exponential time algorithm in Section 3.2.3.

3.2.1 An NP-completeness proof

Theorem 3.1. *The 2-Disjoint Connected Subgraphs problem restricted to instances with $|Z_1| = 2$ is NP-complete.*

Proof. We use a reduction from 3-Satisfiability, which is well-known to be NP-complete (cf. [127]). Let $X = \{x_1, \ldots, x_n\}$ be a set of variables and $C = \{c_1, \ldots, c_m\}$ be a set of clauses forming an instance of 3-Satisfiability. Let $\overline{X} := \{\overline{x} \mid x \in X\}$. We construct a graph G, depicted in Figure 3.1, as follows. Every literal in $X \cup \overline{X}$ and every clause in C is represented by a vertex in G. There is an edge between $x \in X \cup \overline{X}$ and $c \in C$ if and only if x appears in c. For $i = 1, \ldots, n-1$, x_i and $\overline{x}_i$ are adjacent to both x_{i+1} and $\overline{x}_{i+1}$. We add two vertices f_1 and f_2 to G, where f_1 is adjacent to x_1 and $\overline{x}_1$, and f_2 is adjacent to x_n and $\overline{x}_n$.

We claim that the graph G, together with the sets $Z_1 := \{f_1, f_2\}$ and $Z_2 := C$, is a yes-instance of the 2-Disjoint Connected Subgraphs problem if and only if C is satisfiable.

Suppose $t : X \to \{\text{true}, \text{false}\}$ is a satisfying truth assignment for C. Let X_T (respectively X_F) be the set of variables that are set to true (respectively false) by t, and let $\overline{X}_T := \{\overline{x} \mid x \in X_T\}$ and $\overline{X}_F := \{\overline{x} \mid x \in X_F\}$. We denote the set of true and false literals by T and F respectively, i.e., $T := X_T \cup \overline{X}_F$

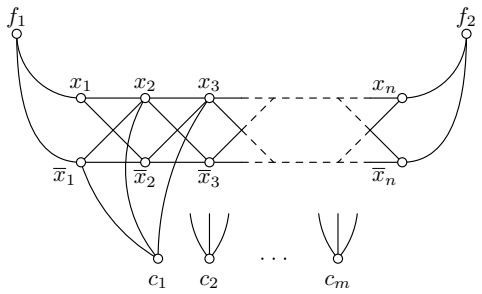

Figure 3.1: The graph G, in case $c_1 = (\overline{x}_1 \vee x_2 \vee x_3)$.

and $F := X_F \cup \overline{X}_T$. Note that exactly one literal of each pair $x_i, \overline{x}_i$ belongs to T, i.e., is set to true by t, and the other one belongs to F. Hence, the vertices in $F \cup \{f_1, f_2\}$ induce a connected subgraph G_1 of G. Since t is a satisfying truth assignment, every clause vertex is adjacent to a vertex in T. Hence the vertices in $T \cup C$ induce a connected subgraph G_2 of G, which is vertex-disjoint from G_1.

To prove the reverse statement, suppose G_1 and G_2 are two vertex-disjoint connected subgraphs of G such that $\{f_1, f_2\} \subseteq V(G_1)$ and $C \subseteq V(G_2)$. Since f_1 and f_2 form an independent set in G and G_1 is connected, at least one of each pair $x_i, \overline{x}_i$ must belong to $V(G_1)$. Since the vertices of C form an independent set in G, every clause vertex must be adjacent to at least one literal vertex in $(X \cup \overline{X}) \cap V(G_2)$. Let t be a truth assignment that sets those literals to true, and their negations to false. For each pair $x_i, \overline{x}_i$ both literals of which belong to $V(G_1)$, t sets exactly one literal to true, and the other one to false. Then t is a satisfying truth assignment for C. □

3.2.2 A complexity classification for P_ℓ-free graphs

In Section 2.3 we proved that a graph G is P_4-free if and only if each connected induced subgraph of G contains a dominating induced C_4 or a dominating vertex (Theorem 2.1). We use this characterization of P_4-free graphs in the proof of the complexity classification of the 2-Disjoint Connected Subgraphs problem below.

Theorem 3.2. *The* 2-Disjoint Connected Subgraphs *problem restricted to the class of P_ℓ-free graphs is polynomial time solvable if $\ell \leq 4$ and* NP*-complete if $\ell \geq 5$.*

Proof. Assume $\ell \leq 4$. Let $G = (V, E)$ be a P_ℓ-free, and consequently P_4-free, graph with non-empty disjoint sets $Z_1, Z_2 \subseteq V$. Suppose G, together with sets Z_1 and Z_2, is a yes-instance of the 2-Disjoint Connected Subgraphs problem, and let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be vertex-disjoint connected subgraphs of G such that $Z_i \subseteq V_i$ for $i = 1, 2$. Note that both G_1 and G_2 are P_4-free. As a result of Theorem 2.1, there exist connected sets D_1, D_2 such that D_i dominates V_i and $|D_i| \in \{1, 4\}$ for $i = 1, 2$. So to check whether G, together with Z_1 and Z_2, is a yes-instance of the 2-Disjoint Connected Subgraphs problem, we act as follows.

We guess a vertex $d_1 \in V \setminus Z_2$. If d_1 does not dominate Z_1, we guess another vertex d_1. If d_1 dominates Z_1, we check if Z_2 is contained in one component G_2 of $G[V \setminus (Z_1 \cup \{d_1\})]$. If so, then $G_1 := G[Z_1 \cup \{d_1\}]$ and G_2 form a solution of the 2-Disjoint Connected Subgraphs problem. Otherwise, we choose another vertex d_1. If we have checked every vertex in $V \setminus Z_2$ without finding a solution, then we guess a 4-tuple $D_1 \subseteq V \setminus Z_2$ and repeat the above procedure with D_1 instead of d_1. If we do not find a solution for any 4-tuple D_1, then (G, Z_1, Z_2) is a no-instance of the 2-Disjoint Connected Subgraphs problem. Since we can perform all checks in polynomial time, this finishes the proof of the polynomial cases.

We now show that the 2-Disjoint Connected Subgraphs problem is NP-complete for P_ℓ-free graphs if $\ell \geq 5$. Clearly, the problem lies in NP. We prove NP-completeness by using a reduction from the NP-complete Hypergraph 2-Colorability problem that asks if a given hypergraph is 2-colorable (cf. [127]). Let $H = (Q, \mathcal{S})$ be a hypergraph with $Q = \{q_1, \ldots, q_n\}$ and $\mathcal{S} = \{S_1, \ldots, S_m\}$. We may assume $m \geq 2$ and $S_i \neq \emptyset$ for each $S_i \in \mathcal{S}$. Let G be the graph obtained from the incidence graph of H by adding the vertices $\mathcal{S}' = \{S'_1, \ldots, S'_m\}$, where $S'_i = S_i$ for every $1 \leq i \leq m$, and by adding the following edges: $q_i S'_j$ if and only if $q_i \in S'_j$, and $q_i q_j$ if and only if $i \neq j$. See Figure 3.2 for the graph G obtained in this way from the hypergraph $(Q, \mathcal{S})$ with $Q = \{q_1, q_2, q_3\}$ and $\mathcal{S} = \{\{q_1, q_3\}, \{q_1, q_2\}, \{q_1, q_2, q_3\}\}$. Note that G is a split graph. Hence G is P_5-free, and consequently G is P_ℓ-free for any $\ell \geq 5$. We claim that G, together with the sets $\mathcal{S}$ and $\mathcal{S}'$, is a yes-instance of the 2-Disjoint Connected Subgraphs problem if and only if $(Q, \mathcal{S})$ has a 2-coloring.

Suppose G_1 and G_2 are vertex-disjoint connected subgraphs of G such

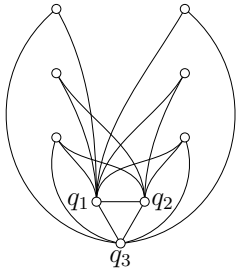

Figure 3.2: The graph G.

that $\mathcal{S} \subseteq V(G_1)$ and $\mathcal{S}' \subseteq V(G_2)$. Without loss of generality, assume that $V_1 := V(G_1)$ and $V_2 := V(G_2)$ form a partition of V. Then there exists a partition (Q_1, Q_2) of Q such that $V_1 = \mathcal{S} \cup Q_1$ and $V_2 = \mathcal{S}' \cup Q_2$. Note that $\mathcal{S}$ is an independent set in G. Hence $Q_1 \neq \emptyset$ and every vertex in $\mathcal{S}$ is adjacent to at least one vertex in Q_1. Similarly, $Q_2 \neq \emptyset$ and every vertex in $\mathcal{S}'$ has at least one neighbor in Q_2. Since $S'_i = S_i$ for every $1 \leq i \leq m$, (Q_1, Q_2) is a 2-coloring of $(Q, \mathcal{S})$.

Now suppose $(Q, \mathcal{S})$ has a 2-coloring (Q_1, Q_2). Then it is clear that $G[\mathcal{S} \cup Q_1]$ and $G[\mathcal{S}' \cup Q_2]$ are connected, so we can choose $G_1 := G[\mathcal{S} \cup Q_1]$ and $G_2 := G[\mathcal{S}' \cup Q_2]$. This finishes the proof of the NP-complete cases. □

3.2.3 An exact algorithm

In this section we present an exact algorithm that solves the 2-Disjoint Connected Subgraphs problem for P_ℓ-free graphs faster than the trivial $\mathcal{O}^*(2^n)$, for every ℓ. We first need to introduce some additional terminology.

Let $G = (V, E)$ be a graph, and let $S \subset V$ and $p, q \in V \setminus S$. We say that p is *separated* from q by S if every path in G from p to q contains a vertex of S. The *distance* $d_G(u, v)$ between two vertices u and v in a graph G is the *length*, $|V(P)| - 1$, of a shortest path P between them. For any vertex $v \in V$ and set $S \subseteq V$, we write $d_G(v, S)$ to denote the length of a shortest path from v to S, i.e., $d_G(v, S) := \min_{w \in S} d_G(v, w)$. The set $N^r_G(S) := \{u \in V \mid d_G(u, S) \leq r\}$ is called the *r-neighborhood* of a set S. A set S *r-dominates* a set S' if $(S' \setminus S) \subseteq N^r_G(S)$; we also say that S r-dominates the graph $G[S']$. A subgraph H of G is an *r-dominating subgraph* of G if $V(H)$ r-dominates G. In case $r = 1$, we use "dominating" instead of "1-dominating". A set $S \subseteq V$ is called a *(k, r)-center* of G if $|S| \leq k$ and

$N_G^r(S) = V$. Recall that a set S is called *connected* if $G[S]$ is connected. The class of graphs all connected induced subgraphs of which have a connected (k, r)-center is denoted by $\mathcal{G}^{k,r}$.

Lemma 3.3. *Let $G = (V, E)$ be a connected induced subgraph of a graph $G' \in \mathcal{G}^{k,r}$. For each subset $Z \subseteq V$, there exists a set $D^* \subseteq V$ with $|D^*| \leq (r-1)|Z| + k$ such that $G[D^* \cup Z]$ is connected.*

Proof. By definition of $\mathcal{G}^{k,r}$, G has a connected (k, r)-center D_0. Let $D_i := \{v \in V \mid d_G(v, D_0) = i\}$ for $i = 1, \ldots, r$. Note that the sets $D_0, \ldots, D_r$ form a partition of V. Let z be any vertex of Z and suppose $z \in D_i$ for some $0 \leq i \leq r$; note that this i is uniquely defined. By definition, there exists a path P^z of length i from z to a vertex in D_0, and it is clear that $D_0 \cup P^z \setminus \{z\}$ is a connected set of size $(i-1) + |D_0|$ that dominates z. Let $\mathcal{P} := \bigcup_{z \in Z} P^z \setminus \{z\}$. Clearly, $D^* := D_0 \cup \mathcal{P}$ is a connected set dominating Z. In the worst case, we have $Z \subseteq D_r$ and all the paths in $\mathcal{P}$ are mutually vertex-disjoint, in which case $|D^*| = (r-1)|Z| + |D_0| \leq (r-1)|Z| + k$. This finishes the proof of Lemma 3.3. □

Lemma 3.3 implies the following.

Corollary 3.4. *For any fixed k, the 2-Disjoint Connected Subgraphs problem for $\mathcal{G}^{k,r}$ can be solved in polynomial time if $r = 1$, or if one of the given sets Z_1 or Z_2 of vertices has fixed size.*

Proof. Let $G = (V, E)$ be a connected graph in $\mathcal{G}^{k,r}$, and let G together with sets $Z_1, Z_2 \subseteq V$ be an instance of the 2-Disjoint Connected Subgraphs problem. If G, together with the sets Z_1 and Z_2, is a yes-instance, then G has two vertex-disjoint connected subgraphs G_1, G_2 such that $Z_i \subseteq V(G_i)$ for $i = 1, 2$. By Lemma 3.3, there exists a set $D^* \subseteq V(G_1)$ such that $|D^*| \leq (r-1)|Z_1| + k$ and $G[D^* \cup Z_1]$ is connected; a similar set exists for Z_2. Note that D^* has fixed size k if $r = 1$, and D^* has fixed size $(r-1)|Z_1| + k$ if Z_1 has fixed size. Hence, we can solve the problem in polynomial time by performing the following procedure.

Initially, set $V_1 := Z_1$ and $V_2 := Z_2$. For all sets $Z' \subseteq V \setminus Z_2$ in order of increasing cardinality up to at most $(r-1)|Z_1| + k$, check whether $G[Z' \cup Z_1]$ is connected. If not, choose another set Z'. Otherwise, add Z' to V_1 and check for every vertex $v \in V \setminus (Z' \cup Z_1 \cup Z_2)$ whether v is separated from

Z_2 by $Z_1 \cup Z'$. If so, put v in V_1, otherwise put v in V_2. After checking all vertices of $V \setminus (Z' \cup Z_1 \cup Z_2)$, verify whether the graph $G[V_2]$ is connected. If so, the graphs $G_1 := G[V_1]$ and $G_2 := G[V_2]$ form the desired solution. If not, choose another set Z' and repeat the procedure. If no solution is found for any set Z', then no solution to the problem exists.

Since all checks can be done in polynomial time and we only have to perform this procedure a fixed number of times, the 2-Disjoint Connected Subgraphs problem for $\mathcal{G}^{k,r}$ can indeed be solved in polynomial time if $r = 1$, or if one of the given sets of vertices has fixed size. □

From now on, we assume that $r \geq 2$ (and that the sets Z_1, Z_2 may have arbitrary size). We present the algorithm `SPLIT` that solves the 2-Disjoint Connected Subgraphs problem for any $G \in \mathcal{G}^{k,r}$, or concludes that a solution does not exist. We assume $1 \leq |Z_1| \leq |Z_2|$ and define $Z := V \setminus (Z_1 \cup Z_2)$. Algorithm `SPLIT` distinguishes between whether or not Z_1 has a "reasonably" small size, i.e., size at most an for some number $0 < a \leq \frac{1}{2(r-1)}$, the value of which will be determined later.

Case 1. $|Z_1| \leq an$.
For all sets $Z' \subseteq Z$ in order of increasing cardinality up to at most $(r-1)|Z_1|+k$, check whether $G_1 := G[Z' \cup Z_1]$ is connected and $G[(Z \setminus Z') \cup Z_2]$ has a component G_2 containing all vertices of Z_2. If so, output G_1 and G_2. If not, choose another set Z' and repeat the procedure. If no solution is found for any set Z', then output No.

Case 2. $|Z_1| > an$.
Perform the procedure described in Case 1 for all sets $Z' \subseteq Z$ in order of increasing cardinality up to at most $\lceil (1-2a)n \rceil$.

Theorem 3.5. *For any fixed k and $r \geq 2$, algorithm `SPLIT` solves the* 2-Disjoint Connected Subgraphs *problem for any n-vertex graph in $\mathcal{G}^{k,r}$ in $\mathcal{O}^*((f(r))^n)$ time, where*

$$f(r) = \min_{0<c\leq 0.5} \left\{ \max \left\{ \frac{1}{c^c(1-c)^{1-c}}, 2^{1-\frac{2c}{r-1}} \right\} \right\}.$$

Proof. Let $G = (V, E)$ be a graph in $\mathcal{G}^{k,r}$ with $|V| = n$, and let $Z_1, Z_2 \subseteq V$ be two non-empty disjoint sets of vertices of G with $1 \leq |Z_1| \leq |Z_2|$. If Case 1 occurs, the correctness of `SPLIT` follows from Lemma 3.3. If Case 2 occurs,

correctness follows from the fact that all subsets of Z may be checked if necessary, as $|Z_1| > an$ implies $|Z_2| > an$, and therefore $|Z| \leq (1-2a)n$. We are left to prove that the running time mentioned in Theorem 3.5 is correct. We consider Case 1 and Case 2.

Case 1. $|Z_1| \leq an$.

In the worst case, the algorithm has to check all sets $Z' \subseteq Z$ in order of increasing cardinality up to $(r-1)|Z_1|+k \leq (r-1)an+k$. Let $c := (r-1)a$, and note that $c \leq \frac{1}{2}$ since we assumed $a \leq \frac{1}{2(r-1)}$. Then we must check at most $\sum_{i=1}^{cn+k} \binom{n}{i}$ sets Z'. Note that

$$\begin{aligned}
\sum_{i=cn+1}^{cn+k} \binom{n}{i} &= \binom{n}{cn+1} + \binom{n}{cn+2} + \ldots + \binom{n}{cn+k} \\
&= \binom{n}{cn} \cdot \sum_{i=1}^{k} \left(\prod_{j=1}^{i} \frac{n-cn-i+1}{cn+i} \right) \\
&\leq \binom{n}{cn} \cdot \sum_{i=1}^{k} \frac{(n-cn)^i}{(cn)^i} \\
&= \binom{n}{cn} \cdot \sum_{i=1}^{k} \left(\frac{1-c}{c}\right)^i .
\end{aligned}$$

This, together with the fact that

$$\sum_{i=1}^{cn} \binom{n}{i} \leq cn \cdot \binom{n}{cn} ,$$

means that the number of sets we have to check is

$$\mathcal{O}\left(cn \cdot \binom{n}{cn} \right) .$$

For each set all the required checks can be done in polynomial time. Since k is a fixed constant, independent of n, we can use Using Stirling's approximation, $n! \approx n^n e^{-n}\sqrt{2\pi n}$, to conclude that the running time for Case 1 is

$$\mathcal{O}^*\left(\left(\frac{1}{c^c \cdot (1-c)^{1-c}} \right)^n \right) .$$

Input graph is...	SPLIT runs in...
P_5-free	$\mathcal{O}^*(1.5790^n)$
P_6-free	$\mathcal{O}^*(1.5790^n)$
P_ℓ-free ($\ell \geq 7$)	$\mathcal{O}^*((f(\ell-3))^n)$
P_7-free	$\mathcal{O}^*(1.7737^n)$
P_8-free	$\mathcal{O}^*(1.8135^n)$
P_{100}-free	$\mathcal{O}^*(1.9873^n)$

Table 3.1: The time complexities of SPLIT for some graph classes.

Case 2. $|Z_1| > an$.
In the worst case, the algorithm has to check all $\mathcal{O}(2^{(1-2a)n})$ sets $Z' \subseteq Z$ in order of increasing cardinality up to $\lceil (1-2a)n \rceil$. Since for each set all the required checks can be done in polynomial time, the running time for Case 2 is

$$\mathcal{O}^*\Big(\Big(2^{1-2a}\Big)^n\Big) = \mathcal{O}^*\Big(\Big(2^{1-\frac{2c}{r-1}}\Big)^n\Big).$$

Since we do not know in advance whether Case 1 or Case 2 will occur, the appropriate value of c can be computed by taking

$$\min_{0<c\leq 0.5}\left\{\max\left\{\frac{1}{c^c\cdot(1-c)^{1-c}}, 2^{1-\frac{2c}{r-1}}\right\}\right\}.$$

This finishes the proof of Theorem 3.5. □

See Table 3.1 for the time complexities of SPLIT for some graph classes. To prove that the time complexities in Table 3.1 are correct, we use the characterization of P_6-free graphs that we obtained in Section 2.4. We repeat the theorem here for ease of reference.

Theorem 2.5. *A graph G is P_6-free if and only if each connected induced subgraph of G contains a dominating induced C_6 or a dominating (not necessarily induced) complete bipartite graph. Moreover, we can find such a dominating subgraph of G in $\mathcal{O}(n^3)$ time.*

Theorem 3.6. *The time complexities of SPLIT shown in Table 3.1 are correct.*

Proof. As we mentioned in Section 2.1, Bacsó and Tuza [16] proved that, for any $\ell \geq 7$, a graph G is P_ℓ-free if and only if each connected induced

subgraph of G has a dominating subgraph of diameter at most $\ell-4$. Since a graph of diameter at most $\ell-4$ has an $(\ell-4)$-dominating vertex, this means that every P_ℓ-free graph is in $\mathcal{G}^{1,\ell-3}$ for any $\ell \geq 7$. Evaluating the function f in Theorem 3.5 at $r=4$, $r=5$ and $r=97$ yields the running times for P_7-free, P_8-free and P_{100}-free graphs in Table 3.1. Since $f(2) \approx 1.5790$, it remains to show that the classes of P_5-free and P_6-free graphs belong to $\mathcal{G}^{k,2}$ for some constant k. Since every induced C_6 has a dominating connected set of size 4, and every complete bipartite graph has a dominating connected set of size 2, the class of P_6-free graphs is in $\mathcal{G}^{4,2}$ as a result of Theorem 2.5. The observation that the class of P_5-free graphs is a subclass of the class of P_6-free graphs finishes the proof of Theorem 3.6. □

Let G be the graph obtained from a complete graph with vertex set $\{x_1,\ldots,x_p\}$ by adding an edge between each x_i and a new vertex y_i, which is only made adjacent to x_i. The graph G is P_5-free, and G does not belong to $\mathcal{G}^{k,1}$ for any constant k. This example shows that we cannot reduce $r=2$ to $r=1$ for P_5-free graphs.

3.3 The Longest Path Contractibility problem

3.3.1 A complexity classification for P_ℓ-free graphs

Before stating the main theorem of this section, we first present a number of useful results.

Theorem 3.7. *The P_4-Contractibility problem is NP-complete for the class of P_6-free graphs.*

Proof. Brouwer and Veldman [52] give an elegant reduction from the Hypergraph 2-Colorability problem to show that the P_4-Contractibility problem is NP-complete. Given a hypergraph $(Q,\mathcal{S})$ they construct a graph G such that $(Q,\mathcal{S})$ has a 2-coloring if and only if G is P_4-contractible. Hence, to prove Theorem 3.7, it suffices to show that G is P_6-free. Below we show how G is constructed.

Let $(Q,\mathcal{S})$ be a hypergraph with $Q=\{q_1,\ldots,q_n\}$ and $\mathcal{S}=\{S_1,\ldots,S_m\}$, and assume without loss of generality that $S_m=Q$. The graph $G=(V,E)$ is constructed from the incidence graph of $(Q,\mathcal{S})$ as follows. First we add

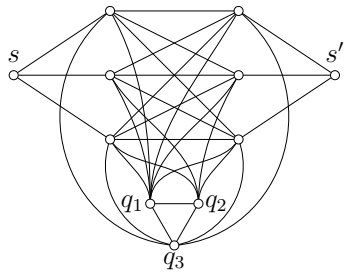

Figure 3.3: The graph G.

two new vertices s, s' and a copy $\mathcal{S}' = \{S'_1, \ldots, S'_m\}$ of $\mathcal{S}$, such that $S'_i = S_i$ for every $1 \leq i \leq m$. Then we add the following edges:

- $S_i S'_j$ for every $1 \leq i, j \leq m$;
- sS_i for every $1 \leq i \leq m$;
- $s'S'_i$ for every $1 \leq i \leq m$;
- $S'_i q_j$ if and only if $q_j \in S_i$;
- $q_i q_j$ if and only if $i \neq j$.

See Figure 3.3 for the graph G obtained in this way from the hypergraph $(Q, \mathcal{S})$ with $Q = \{q_1, q_2, q_3\}$ and $\mathcal{S} = \{\{q_1, q_3\}, \{q_1, q_2\}, \{q_1, q_2, q_3\}\}$. We claim that G is P_6-free. This can be seen as follows. Let P be an induced path of G with maximum length over all induced paths of G. Note that P contains at most 2 vertices of Q, since Q is a clique in G. Suppose P starts in s or s'. By symmetry we may assume that P starts in s. Let S_i be the next vertex of P. If P does not contain any vertex of $\mathcal{S}'$, then $V(P) \setminus \{s, S_i\} \subseteq Q$ and P has length at most 3. Suppose P contains some vertex S'_j. Then $sS_iS'_j$ is a subpath of P and the next vertex on P is either s' or lies in Q. In the first case $P = sS_iS'_js'$, so P has length 3. In the second case P ends in Q (as $G[\mathcal{S} \cup \mathcal{S}']$ is complete bipartite) and has length at most 4.

Suppose P starts in S_i or S'_i for some $1 \leq i \leq m$ and does not end in s or s'. By symmetry we may assume P starts in S_i. If the second vertex of P is s, then P does not contain any vertex of $\mathcal{S}'$ and has length at most 4. If the second vertex of P is from Q, then P does not contain a vertex from $\mathcal{S}'$. In that case, P either ends in Q and has length at most 2, or P ends in $\mathcal{S}$ and consequently does not contain s or more than two vertices of Q, so P has length at most 3. If the second vertex of P is from $\mathcal{S}'$ and P does not end

in this vertex, then P ends in Q and has length at most 3. Note that we do not have to consider the case where P ends in $\mathcal{S}$, as we already considered that case before.

Suppose P starts in Q and does not end in a vertex in $\{s, s'\} \cup \mathcal{S} \cup \mathcal{S}'$. Then P ends in Q, and consequently, P has length at most 1. We conclude that G is indeed P_6-free. □

A pair of vertices (u, v) of a graph G is *P_ℓ-suitable* for some integer $\ell \geq 3$ if and only if G has a P_ℓ-witness structure $\mathcal{W}$ with $W(p_1) = \{u\}$ and $W(p_\ell) = \{v\}$, where $P_\ell = p_1 \cdots p_\ell$. The two vertices making up the outer witness sets of the right P_4-witness structure in Figure 1.1 in Section 1.1.3 form a P_4-suitable pair.

Lemma 3.8. *For $\ell \geq 3$, a graph is P_ℓ-contractible if and only if it has a P_ℓ-suitable pair.*

Proof. By definition, a graph G is P_ℓ-contractible if G has a P_ℓ-suitable pair of vertices. To prove the reverse statement, let G be a P_ℓ-contractible graph and let $\mathcal{W}$ be a P_ℓ-witness structure of G. Suppose $|W(p_1)| \geq 2$. Let $x \in W(p_1)$ be a vertex that is not a cut vertex of $G[W(p_1)]$; note that any graph on at least two vertices has at least two such vertices, namely the leaves of any spanning tree.

Suppose $W(p_1)$ contains a vertex $y \neq x$ adjacent to $W(p_2)$. Then we define $W'(p_1) := \{x\}$, $W'(p_2) := W(p_2) \cup (W(p_1) \setminus \{x\})$ and $W'(p_i) := W(p_i)$ for $i = 3, \ldots, \ell$.

Suppose x is the only vertex of $W(p_1)$ adjacent to $W(p_2)$. As $|W(p_1)| \geq 2$ and $G[W(p_1)]$ is connected, there exists a vertex $y \in W(p_1) \setminus \{x\}$ that is not a cut vertex of $G[W(p_1)]$. We define $W'(p_1) := \{y\}$, $W'(p_2) := W(p_2) \cup (W(p_1) \setminus \{y\})$ and $W'(p_i) := W(p_i)$ for $i = 3, \ldots, \ell$. So given a P_ℓ-witness structure $\mathcal{W}$ of G, we can always find a P_ℓ-witness structure $\mathcal{W}'$ of G with $|W'(p_1)| = 1$. Since $\ell \geq 3$, we did not change the witness sets $W(p_\ell)$ and $W(p_{\ell-1})$ in obtaining $\mathcal{W}'$. Hence, we can repeat the arguments above for $W(p_\ell)$ to obtain a P_ℓ-witness structure $\mathcal{W}''$ of G with $|W''(p_1)| = |W''(p_\ell)| = 1$. By definition, the two vertices of $W''(p_1) \cup W''(p_\ell)$ form a P_ℓ-suitable pair of G. □

Lemma 3.9. *Let x and y be two neighbors of a vertex u in a graph G with $xy \in E(G)$, and let v be some other vertex in G. Then (u, v) is a P_ℓ-suitable pair of G if and only if (u, v) is a P_ℓ-suitable pair of $G \setminus xy$.*

Proof. Suppose (u, v) is a P_ℓ-suitable pair of G. By definition, G has a P_ℓ-witness structure $\mathcal{W}$ with $W(p_1) = \{u\}$ and $W(p_\ell) = \{v\}$. Then $x, y \in N(u)$ are both in the same witness set, namely $W(p_2)$. Hence we may contract edge xy in order to obtain a P_ℓ-witness structure $\mathcal{W}'$ for $G \setminus xy$ with $W'(p_1) = \{u\}$ and $W'(p_\ell) = \{v\}$. The reverse implication is trivial. □

Lemma 3.10. *For any edge xy of a P_ℓ-free graph G, the graph $G \setminus xy$ is P_ℓ-free.*

Proof. Let $G = (V, E)$ be a P_ℓ-free graph, and let z be the vertex that is being created by contracting the edge $xy \in E$. Suppose $G \setminus xy$ is not P_ℓ-free and let $p_1 p_2 \cdots p_\ell$ be an induced P_ℓ in $G \setminus xy$. Since G is P_ℓ-free, we must have $z = p_j$ for some $2 \leq j \leq \ell - 1$. Suppose x is adjacent to both p_{j-1} and p_{j+1} in G. Then the path $p_1 \cdots p_{j-1} x p_{j+1} \cdots p_\ell$ forms an induced P_ℓ in G, a contradiction. Therefore x, and by symmetry y, cannot be adjacent to both p_{j-1} and p_{j+1} in G. Without loss of generality, assume that $p_{j-1}x \in E$ and $yp_{j+1} \in E$. Then the path $p_1 p_2 \cdots p_{j-1} x y p_{j+1} \cdots p_{\ell-1}$ forms an induced P_ℓ in G, contradicting the P_ℓ-freeness of G. □

We now present a polynomial time algorithm for deciding whether a P_5-free graph is P_4-contractible.

Theorem 3.11. *The P_4-Contractibility problem is solvable in polynomial time for the class of P_5-free graphs.*

Proof. Let $G = (V, E)$ be a connected P_5-free graph. Lemma 3.8 states that G is P_4-contractible if and only if G contains a P_4-suitable pair (u, v). Since G has $\mathcal{O}(|V|^2)$ pairs (u, v), it suffices to show that we can check in polynomial time whether a given pair (u, v) is P_4-suitable. It follows from the definition of a P_4-witness structure and the P_5-freeness of G that we only need to consider pairs of vertices at distance 3. If there does not exist such a pair, then G is not P_4-contractible. Suppose (u, v) is a pair of vertices of G with $d_G(u, v) = 3$.

Claim 1. We may without loss of generality assume that $N(u)$ and $N(v)$ are independent sets of cardinality at least 2.

We prove Claim 1 as follows. Lemma 3.9 and Lemma 3.10 together immediately imply that we may assume $N(u)$ and $N(v)$ to be independent sets. Now suppose that $N(u)$ has cardinality 1, say $N(u) = \{x\}$. It is clear that (u, v) is a P_4-suitable pair of G if and only if $N(v)$ is contained in one component of $G[V \setminus \{u, v, x\}]$, which can be checked in polynomial time. Hence we may assume that $|N(u)| \geq 2$, and by symmetry $|N(v)| \geq 2$.

Claim 2. Let x and x' be two vertices of G such that x is adjacent to a vertex $w \in N(u)$ but not to a vertex $w' \in N(u)$, and x' is adjacent to w' but not to w. Then $N(u) \subseteq N(x) \cup N(x')$.

We prove Claim 2 as follows. Clearly $u \notin \{x, x'\}$. As $N(u)$ is an independent set by Claim 1, u is neither adjacent to x nor to x'. Then $xx' \in E$, since otherwise the path $x'w'uwx$ is an induced P_5 as a result of Claim 1, contradicting the P_5-freeness of G. Now suppose there exists a vertex $w'' \in N(u)$ not in $N(x) \cup N(x')$. Since w' and w'' are not adjacent as a result of Claim 1, the path $w''uw'x'x$ is an induced P_5 in G. This contradiction proves Claim 2.

Claim 3. Suppose G has a P_4-witness structure $\mathcal{W}$ with $W(p_1) = \{u\}$ and $W(p_4) = \{v\}$. Then at least one of the following holds:

1. *there exists a vertex $x \in W(p_2) \setminus N(u)$ with $N(u) \subseteq N(x)$;*
2. *there exist vertices $x, x' \in W(p_2) \setminus N(u)$ with $N(u) \subseteq N(x) \cup N(x')$.*

We prove this claim as follows. Suppose $\mathcal{W}$ is a P_4-witness structure of G with $W(p_1) = \{u\}$ and $W(p_4) = \{v\}$, and suppose condition 1 does not hold. We show that condition 2 must hold. By Claim 1, $N(u)$ is an independent set of G containing at least two vertices. Since $N(u) \subseteq W(p_2)$ and $G[W(p_2)]$ is connected, we know that $W(p_2) \setminus N(u) \neq \emptyset$. Let $x \in W(p_2) \setminus N(u)$ be a vertex such that $|N(u) \cap N(x)|$ is maximum over all vertices in $W(p_2) \setminus N(u)$. Since condition 1 does not hold, there exists a vertex $w' \in N(u)$ that is not adjacent to x. Then w' is adjacent to a vertex $x' \in W(p_2) \setminus (N(u) \cup \{x\})$, as otherwise w' would be an isolated vertex in $G[W(p_2)]$. By choice of x, there exists a vertex $w \in N(u) \cap N(x)$ not adjacent to x'. By Claim 2, $N(u) \subseteq N(x) \cup N(x')$. This finishes the proof of Claim 3.

It remains to prove how we can check in polynomial time whether (u, v) is a P_4-suitable pair of G. If (u, v) is a P_4-suitable pair of G, then by definition G has a P_4-witness structure $\mathcal{W}$ with $W(p_1) = \{u\}$ and $W(p_4) = \{v\}$. Any such witness structure satisfies at least one of the two conditions in Claim 3. We can check in polynomial time if these conditions hold after guessing one vertex (respectively two vertices) in $V \setminus (N(u) \cup N(v) \cup \{u, v\})$. If so, we check in polynomial time if $N(v)$ is contained in one component of the remaining graph (without vertex v). If all our guesses are negative, then (u, v) is not a P_4-suitable pair of G. □

Theorem 3.7 and Theorem 3.11 together yield the main result of this section.

Theorem 3.12. *The* Longest Path Contractibility *problem restricted to the class of P_ℓ-free graphs is polynomial time solvable if $\ell \leq 5$ and* NP*-hard if $\ell \geq 6$.*

Proof. First assume $\ell = 5$. Let $G = (V, E)$ be a P_5-free graph. By definition, $\vartheta(G) = 0$ if and only if G is disconnected. Suppose G is connected. Since G does not contain an induced path on more than four vertices, G is clearly not contractible to such a path. Hence we have $\vartheta(G) \leq 4$. By Theorem 3.11, we can check in polynomial time whether G is P_4-contractible. If so, then $\vartheta(G) = 4$. Otherwise, we check if G has a P_3-suitable pair. This is a necessary and sufficient condition for P_3-contractibility according to Lemma 3.8. We can perform this check in polynomial time, since two vertices u, v form a P_3-suitable pair of G if and only if u and v are non-adjacent and $G[V \setminus \{u, v\}]$ is connected. If G is P_3-contractible, then $\vartheta(G) = 3$. If G is not P_3-contractible, then we conclude that $\vartheta(G) = 2$ if G has at least two vertices, and $\vartheta(G) = 1$ otherwise.

Now assume $\ell = 6$. Since a graph G is P_4-contractible if and only if $\vartheta(G) \geq 4$ and the P_4-Contractibility problem is NP-complete for P_6-free graphs by Theorem 3.7, the Longest Path Contractibility problem is NP-hard for P_6-free graphs.

The claim for all other values of ℓ immediately follows from the fact that the class of P_ℓ-free graphs is a subclass of the class of $P_{\ell'}$-free graphs whenever $\ell \leq \ell'$. □

3.3.2 An exact algorithm

Algorithm SPLIT can be extended to an algorithm that solves the Longest Path Contractibility problem on P_6-free graphs in $\mathcal{O}^*(1.5790^n)$ time. This extension is described in detail in the proof of the following theorem.

Theorem 3.13. *The* Longest Path Contractibility *problem for P_6-free graphs can be solved in $\mathcal{O}^*(1.5790^n)$ time.*

Proof. Let $G = (V, E)$ be a P_6-free graph with $|V| = n$. By definition, $\vartheta(G) = 0$ if and only if G is disconnected. Suppose G is connected. Since G does not contain an induced path on six vertices, G is clearly not P_6-contractible. Hence $\vartheta(G) \leq 5$. We first show how we can determine in $\mathcal{O}^*(1.5790^n)$ time if $\vartheta(G) = 5$, i.e., if G is P_5-contractible. We do this by modifying the algorithm SPLIT such that it decides in $\mathcal{O}^*(1.5790^n)$ time whether a pair (u, v) of vertices of G is P_5-suitable. Note that G has $\mathcal{O}(n^2)$ pairs (u, v) and G is P_5-contractible if and only if G has a P_5-suitable pair (u, v) by Lemma 3.8. Before we present the modified algorithm, we introduce some additional terminology and prove a useful claim below.

Let u, v be two vertices of G for which we want to decide if they form a P_5-suitable pair. It follows from the definition of a P_5-witness structure and the P_6-freeness of G that we may without loss of generality assume $d_G(u, v) = 4$. We define the *set of midpoints for* (u, v) as $S(u, v) := \{x \in V \mid d_G(x, u) = d_G(x, v) = 2\}$. If no confusion is possible, we write $S = S(u, v)$. We define two sets T_1 and T_2 as follows. Set $T_1 = T_1(u, v)$ consists of all vertices in $V \setminus (\{u, v\} \cup N(u) \cup N(v) \cup S)$ that are separated from v by S but are not separated from u by S. Set $T_2 = T_2(u, v)$ consists of all vertices in $V \setminus (\{u, v\} \cup N(u) \cup N(v) \cup S)$ that are separated from u by S but are not separated from v by S. Note that $T_1 \cap T_2 = \emptyset$ and that we can obtain these two sets in polynomial time.

Claim 1. We may without loss of generality assume that $V = \{u, v\} \cup N(u) \cup N(v) \cup S \cup T_1 \cup T_2$.

We prove Claim 1 as follows. Suppose $V' = V \setminus (\{u, v\} \cup N(u) \cup N(v) \cup S \cup T_1 \cup T_2)$ is non-empty. By definition of T_1 and T_2, $V' = W_1 \cup W_2$, where W_1 consists of all vertices that are separated from both u and v by S, and W_2 consists of all vertices that are separated from neither u nor v by S.

First suppose $W_1 \neq \emptyset$. Let $x \in W_1$. Note that $S \subseteq W(p_3)$ for any P_5-witness structure $\mathcal{W}$ of G with $W(p_1) = \{u\}$ and $W(p_5) = \{v\}$. Since x is separated from both u and v by S, we must have $x \in W(p_3)$ for any P_5-witness structure $\mathcal{W}$ of G with $W(p_1) = \{u\}$ and $W(p_5) = \{v\}$; otherwise x would be an isolated vertex in $G[W(p_2)]$ or $G[W(p_4)]$, a contradiction. Hence we may contract x with any of its neighbors, which are either in S or which are also separated from both u and v by S. Then (u, v) is P_5-suitable for the resulting (smaller) graph G' if and only if (u, v) is P_5-suitable for G. Furthermore, by Lemma 3.10, G is P_6-free. Hence we may continue with G'.

Now suppose $W_2 \neq \emptyset$. Let P be a shortest path in G from a vertex in $N(u)$ to a vertex in $N(v)$, containing a vertex in W_2 but not containing any vertex of S (such a path exists, since $W_2 \neq \emptyset$). Then P contains at most one vertex $u' \in N(u)$ and at most one vertex $v' \in N(v)$, as otherwise we can replace P by a shorter path. Consequently, P contains neither u nor v, and we may without loss of generality assume that P starts in u' and ends in v'. Let $x \in W_2 \cap V(P)$. If $V(P) = \{u', x, v'\}$, then $d_G(x, u) = d_G(x, v) = 2$. This would mean $x \in S$, a contradiction. Hence P contains another vertex $y \notin \{u', x, v'\}$. Then the path $uu'\overrightarrow{P}v'v$ contains at least six vertices. As G is P_6-free, P is not an induced path in G. Hence, $G[V(P)]$ contains an edge $st \notin E(P)$, where we assume that s occurs before t on the path P from u' to v'. Since $d_G(u, v) = 4$, we have $u'v' \notin E$. This means that at least one of the two vertices s, t is different from u' and v'. We assume without loss of generality that this vertex is s. Then the path $u'\overrightarrow{P}st\overrightarrow{P}v'$ satisfies the requirements but is shorter than P, a contradiction. This proves Claim 1.

We now show how we can modify the algorithm `SPLIT` to determine if (u, v) is a P_5-suitable pair of G. The modified algorithm takes as input the graph $G[V \setminus \{u, v\}]$ with sets $N(u), N(v), S$. It returns Yes if G has three connected subgraphs G_1, G_2, G_3 such that $N(u) \subseteq V(G_1)$, $S \subseteq V(G_2)$ and $N(v) \subseteq V(G_3)$, and it returns No otherwise. The modified algorithm first determines which of the two sets $N(u) \cup N(v)$ and S is the smallest. Since G is P_6-free, we know that $G \in \mathcal{G}^{4,2}$ (see the proof of Theorem 3.6). Like the original algorithm `SPLIT` for graphs in $\mathcal{G}^{4,2}$, the modified algorithm then distinguishes between whether or not this smallest set has a "reasonably" small size, i.e., size at most an for some number $0 < a \leq \frac{1}{2}$, the value of which will be determined later.

First assume $|N(u)| + |N(v)| \leq |S|$. We distinguish two cases.

Case 1. $|N(u)| + |N(v)| \leq an$.
For all sets $Z' \subseteq T_1 \cup T_2$ in order of increasing cardinality up to at most $|N(u)|+|N(v)|+4$, check if $G_1 := G[(Z' \cap T_1) \cup N(u)]$ and $G_3 := G[(Z' \cap T_2) \cup N(v)]$ are both connected. If not, choose another set Z'. Otherwise, check whether S is contained in one component G_2 of the graph $G[(S \cup T_1 \cup T_2) \setminus Z']$. If so, conclude that (u, v) is P_5-suitable. If not, choose another set Z' and repeat the procedure. If no solution is found for any set Z', then conclude that (u, v) is not a P_5-suitable pair of G.

Case 2. $|N(u)| + |N(v)| > an$.
Perform the procedure described in Case 1 for all sets $Z' \subseteq T_1 \cup T_2$ in order of increasing cardinality up to at most $\lceil (1 - 2a)n \rceil$.

Now assume $|S| \leq |N(u)| + |N(v)|$. Again, we distinguish two cases.

Case 1. $|S| \leq an$.
For all sets $Z' \subseteq T_1 \cup T_2$ in order of increasing cardinality up to at most $|S|+4$, check if the graph $G_2 := G[Z' \cup S]$ is connected. If not, choose another set Z'. Otherwise, check whether the graph $G[(N(u) \cup N(v) \cup T_1 \cup T_2) \setminus Z']$ contains two components G_1, G_3 such that $N(u) \subseteq V(G_1)$ and $N(v) \subseteq V(G_3)$. If so, conclude that (u, v) is P_5-suitable. If not, choose another set Z' and repeat the procedure. If no solution is found for any set Z', then conclude that (u, v) is not a P_5-suitable pair of G.

Case 2. $|S| > an$.
Perform the procedure described in Case 1 for all sets $Z' \subseteq T_1 \cup T_2$ in order of increasing cardinality up to at most $\lceil (1 - 2a)n \rceil$.

The proof of correctness and the running time analysis are similar to the proof of Theorem 3.5. Recall that $G \in \mathcal{G}^{4,2}$. Hence we find that $a \approx 0.17054$ is optimal. After checking $\mathcal{O}(n^2)$ pairs of vertices in G on P_5-suitability, we find in $\mathcal{O}^*(1.5790^n)$ time whether G is P_5-contractible or not. If G is P_5-contractible, then $\vartheta(G) = 5$.

Suppose G is not P_5-contractible. We check if G is P_4-contractible. Recall that G is P_4-contractible if and only if G has a P_4-suitable pair (u, v) by Lemma 3.8. Let $u, v \in V$ be a pair of vertices of G. By Lemma 3.8, the definition of a P_4-suitable pair, and the P_4-freeness of G, we may assume

$d_G(u,v) = 3$. Define $Z_1 := N_G(u)$, $Z_2 := N_G(v)$ and $G' := G[V \setminus \{u,v\}]$. Note that $Z_1 \cap Z_2 = \emptyset$ as $d_G(u,v) = 3$. Furthermore G' is P_6-free as G is P_6-free. Hence (G', Z_1, Z_2) is an instance of the 2-Disjoint Connected Subgraphs problem for P_6-free graphs. By Theorem 3.6, we can decide in $\mathcal{O}^*(1.5790^n)$ time whether there exist vertex-disjoint subgraphs G_1, G_2 of G such that $Z_i \subseteq V(G_i)$ for $i = 1, 2$. It is clear that such subgraphs exist if and only if (u,v) is a P_4-suitable pair of G. Since we have to check $\mathcal{O}(n^2)$ pairs (u,v), we can check in $\mathcal{O}^*(1.5790^n)$ time whether or not G is P_4-contractible. If so, then $\vartheta(G) = 4$.

Suppose G is not P_4-contractible. We check if G has a P_3-suitable pair. This is a necessary and sufficient condition for P_3-contractibility according to Lemma 3.8. We can perform this check in polynomial time, since two vertices u, v form a P_3-suitable pair of G if and only if u, v are non-adjacent and $G[V \setminus \{u,v\}]$ is connected. If G is P_3-contractible, then $\vartheta(G) = 3$. If G is not P_3-contractible, then we conclude that $\vartheta(G) = 2$ if G has at least two vertices, and $\vartheta(G) = 1$ otherwise. □

3.4 Conclusion

We showed that the 2-Disjoint Connected Subgraphs problem is already NP-complete if one of the given sets of vertices has cardinality 2. We also showed that the 2-Disjoint Connected Subgraphs problem for the class of P_ℓ-free graphs jumps from being polynomial time solvable to being NP-hard at $\ell = 5$. The Longest Path Contractibility problem restricted to the class of P_ℓ-free graphs can be solved in polynomial time if $\ell \leq 5$, and is NP-complete if $\ell \geq 6$.

Our algorithm `SPLIT` solves the 2-Disjoint Connected Subgraphs problem for P_ℓ-free graphs faster than $\mathcal{O}^*(2^n)$ for any ℓ. We do not know yet how to improve its running time for P_5-free and P_6-free graphs (which are in $\mathcal{G}^{1,2}$ and $\mathcal{G}^{4,2}$, respectively) but expect we can do better for P_ℓ-free graphs with $\ell \geq 7$ (by using a radius argument). The modification of `SPLIT` solves the Longest Path Contractibility problem for P_6-free graphs in $\mathcal{O}^*(1.5790^n)$ time. Furthermore, `SPLIT` might be modified into an exact algorithm that solves the Longest Path Contractibility problem for P_ℓ-free graphs with $\ell \geq 7$ as well. A more interesting question however is to find

fast exact algorithms for solving the 2-Disjoint Connected Subgraphs problem and the Longest Path Contractibility problem for general graphs.

Chapter 4

On graph contractions and induced minors

Most of the results of this chapter appeared in the following paper.

[156] P. van 't Hof, M. Kamiński, D. Paulusma, S. Szeider, and D.M. Thilikos. On contracting graphs to fixed pattern graphs. In: *Proceedings of the 36th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2010)*, volume 5901 of *Lecture Notes in Computer Science*, pages 503–514, Springer, 2010.

Asking whether a graph G contains a graph H as a minor is equivalent to asking whether H can be obtained from a subgraph of G by a sequence of edge contractions. We mentioned at the beginning of Chapter 3 that Robertson and Seymour [232] proved that this problem can be solved in polynomial time for every fixed graph H. Deciding whether a graph H can be obtained from an *induced* subgraph of G, or from G itself, by contracting edges only, an alternative way of formulating the H-Induced Minor Containment problem and the H-Contractibility problem, respectively, seems to be harder; for both problems pattern graphs H are known for which they are NP-complete [52, 103]. For neither problem a complete complexity classification is known. In this chapter we study the computational complexity of these two problems. Using a deep result by Robertson and Seymour [233], proving Wagner's Conjecture, we show that the H-Induced Minor Containment problem can be solved in polynomial time on any non-trivial minor-closed

graph class if H is planar. We also identify polynomial time solvable and NP-complete cases of the H-Contractibility problem and a new variant of this problem. The presence of a dominating vertex in the target graph H seems to play an interesting role in these results.

4.1 Background and results

The following well-known result by Robertson and Seymour [232] was already mentioned in Chapter 3.

Theorem 4.1 ([232]). *For every fixed graph H, the H-Minor Containment problem can be solved in polynomial time.*

It is highly unlikely that a similar result holds for the H-Induced Minor Containment problem, as Fellows, Kratochvíl, Middendorf, and Pfeiffer [103] identify both polynomial time solvable and NP-complete cases of the H-Induced Minor Containment problem. They also prove the following.

Theorem 4.2 ([103]). *For every fixed planar graph H, the H-Induced Minor Containment problem can be solved in polynomial time on planar input graphs.*

In Section 4.2, after recalling some basic notions in parameterized complexity, we consider the Induced Minor Containment problem that takes as input two graphs G and H and asks whether G has H as an induced minor. We show that this problem is fixed parameter tractable in $|V(H)|$ if H is planar and G belongs to any non-trivial minor-closed graph class $\mathcal{G}$, i.e., to a class $\mathcal{G}$ that contains every minor of each of its members, but does not contain all graphs. This result generalizes Theorem 4.2, as the class of planar graphs is minor-closed.

Brouwer and Veldman [52] initiated the research on the H-Contractibility problem. Their main result is stated below.

Theorem 4.3 ([52]). *Let H be a connected triangle-free graph. The H-Contractibility problem is solvable in polynomial time if H has a dominating vertex, and is NP-complete otherwise.*

Note that a connected triangle-free graph with a dominating vertex is a star and that $H = P_4$ and $H = C_4$ are the smallest graphs H for which

H-Contractibility is NP-complete. The research of Brouwer and Veldman [52] was continued by Levin et al. [193, 194].

Theorem 4.4 ([193, 194]). *Let H be a connected graph on at most five vertices. The H-Contractibility problem is solvable in polynomial time if H has a dominating vertex, and is NP-complete otherwise.*

In the papers by Brouwer and Veldman [52] and by Levin et al. [193] several other results are shown. To discuss these we need some extra terminology, which we will use later in the chapter as well. For two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ with $V_1 \cap V_2 = \emptyset$, we denote their *join* by $G_1 \bowtie G_2 = (V_1 \cup V_2, E_1 \cup E_2 \cup \{uv \mid u \in V_1, v \in V_2\})$, and their *disjoint union* by $G_1 \cup G_2 = (V_1 \cup V_2, E_1 \cup E_2)$. For the disjoint union $G \cup G \cup \cdots \cup G$ of k copies of the graph G, we write kG; for $k = 0$ this yields the empty graph $(\emptyset, \emptyset)$. For integers $a_1, a_2, \ldots, a_k \geq 0$, we let $H_i^*(a_1, a_2, \ldots, a_k)$ be the graph $K_i \bowtie (a_1 P_1 \cup a_2 P_2 \cup \cdots \cup a_k P_k)$, where K_i is the complete graph on i vertices and P_i is the path on i vertices. Note that $H_1^*(a_1)$ denotes a star on $a_1 + 1$ vertices.

Brouwer and Veldman [52] show that H-Contractibility is polynomial time solvable for $H = H_1^*(a_1)$ or $H = H_1^*(a_1, a_2)$ for any $a_1, a_2 \geq 0$. These results have been generalized in [193] leading to the following theorem. Observe that $H_i^*(0) = H_{i-1}^*(1) = K_i$ and that K_i-Contractibility is equivalent to K_i-Minor Containment, and hence solvable in polynomial time by Theorem 4.1.

Theorem 4.5 ([193]). *The H-Contractibility problem is solvable in polynomial time for:*

1. *$H = H_1^*(a_1, a_2, \ldots, a_k)$ for any $k \geq 1$ and $a_1, a_2, \ldots, a_k \geq 0$;*
2. *$H = H_2^*(a_1, a_2)$ for any $a_1, a_2 \geq 0$;*
3. *$H = H_3^*(a_1)$ for any $a_1 \geq 0$;*
4. *$H = H_i^*(a_1)$, for any $i \geq 1$ and $0 \leq a_1 \leq 1$.*

The presence of a dominating vertex seems to play an interesting role in the complexity classification of the H-Contractibility problem. So far, in all polynomial time solvable cases of this problem the pattern graph H has a dominating vertex, and in all NP-complete cases H does not have such a vertex. Following this trend, we extend Theorem 4.5 in Section 4.3.1 by

showing that $H_4^*(a_1)$-Contractibility is polynomial time solvable for all $a_1 \geq 0$. In Section 4.3.2 however we present the first class of graphs H with a dominating vertex for which H-Contractibility is NP-complete.

In Section 4.4 we study the following problem, where vertex v specified in the pair (H, v) refers to a fixed vertex v of H.

(H, v)-Contractibility
Instance: A graph G and a positive integer k.
Question: Does G have an H-witness structure $\mathcal{W}$ with $|W(v)| \geq k$?

We show that (H, v)-Contractibility is NP-complete whenever H is connected and v is not a dominating vertex of H. For example, let $P_3 = p_1p_2p_3$. Then the (P_3, p_3)-Contractibility problem is NP-complete (whereas P_3-Contractibility is polynomial time solvable).

4.2 Induced minors in minor-closed graph classes

We start this section with a short introduction on the complexity classes XP and FPT. Both classes are defined in the framework of parameterized complexity as developed by Downey and Fellows [95]. The complexity class XP consists of parameterized decision problems Π such that for each instance (I, k) it can be decided in $\mathcal{O}(f(k) \cdot |I|^{g(k)})$ time whether $(I, k) \in \Pi$, where f and g are computable functions depending only on k. So XP consists of parameterized decision problems which can be solved in polynomial time if the parameter is considered to be a constant. A problem is *fixed parameter tractable* in k if an instance (I, k) can be solved in time $\mathcal{O}(f(k) \cdot |I|^c)$, where f denotes a computable function and c a constant independent of k. Therefore, such an algorithm may provide a solution to the problem efficiently if the parameter is reasonably small. The complexity class FPT $\subseteq$ XP is the class of all fixed-parameter tractable decision problems.

A graph class is called *non-trivial* if it does not contain all graphs. We show that the Induced Minor Containment problem is fixed parameter tractable in $|V(H)|$ on input pairs (G, H) with G from any fixed non-trivial minor-closed graph class $\mathcal{G}$ and H planar. Before doing this we first recall the following notions. A *tree decomposition* of a graph $G = (V, E)$ is a pair $(\mathcal{X}, T)$, where $\mathcal{X} = \{X_1, \ldots, X_r\}$ is a collection of *bags*, which are subsets of

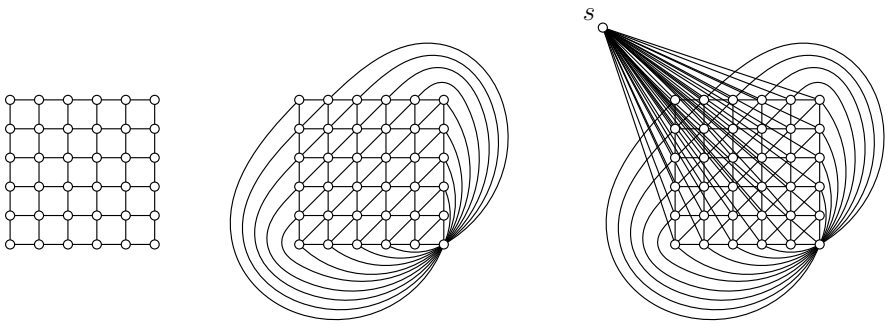

Figure 4.1: The graphs M_6, Γ_6, and Π_6, respectively.

V, and T is a tree on vertex set $\mathcal{X}$ with the following three properties. First, $\bigcup_{i=1}^{r} X_i = V$. Second, for each $uv \in E$, there exists a bag X_i such that $\{u, v\} \subseteq X_i$. Third, if $v \in X_i$ and $v \in X_j$ then all bags in T on the (unique) path between X_i and X_j contain v. The *width* of a tree decomposition $(\mathcal{X}, T)$ is $\max\{|X_i| - 1 \mid i = 1, \ldots, r\}$, and the *treewidth* $\mathrm{tw}(G)$ of G is the minimum width over all possible tree decompositions of G.

Our proof idea is as follows. We check if the input graph G has sufficiently large treewidth. If not, then we apply the monadic second-order logic result of Courcelle [84]. Otherwise, we show that G always contains H as an induced minor. Before going into details, we first introduce some additional terminology.

The $k \times k$ *grid* M_k has as vertex set all pairs (i, j) for $i, j = 0, 1, \ldots, k - 1$, and two vertices (i, j) and (i', j') are joined by an edge if and only if $|i - i'| + |j - j'| = 1$. For $k \geq 2$, let Γ_k denote the graph obtained from M_k by triangulating its faces as follows: add an edge between vertices (i, j) and (i', j') if $i - i' = 1$ and $j' - j = 1$, and add an edge between corner vertex $(k - 1, k - 1)$ and every external vertex that is not already adjacent to $(k - 1, k - 1)$, i.e., every vertex (i, j) with $i \in \{0, k - 1\}$ or $j \in \{0, k - 1\}$, apart from the vertices $(k - 2, k - 1)$ and $(k - 1, k - 2)$. We let Π_k denote the graph obtained from Γ_k by adding a new vertex s that is adjacent to every vertex of Γ_k. See Figure 4.1 for the graphs M_6, Γ_6, and Π_6.

Let $\mathcal{F}$ denote a set of graphs. Then a graph G is called *$\mathcal{F}$-minor-free* if G does not contain a graph in $\mathcal{F}$ as a minor. If $\mathcal{F} = \{F\}$ we say that G is *F-minor-free*. We need the following results by Fomin et al. [112] and by Fellows et al. [103], respectively.

Theorem 4.6 ([112]). *For every graph F, there is a constant c_F such that every connected F-minor-free graph of treewidth at least $c_F \cdot k^2$ is Γ_k-contractible or Π_k-contractible.*

Theorem 4.7 ([103]). *For every planar graph H, there is a constant b_H such that every planar graph of treewidth at least b_H contains H as an induced minor.*

We also recall the well-known result of Robertson and Seymour [233] proving Wagner's conjecture.

Theorem 4.8 ([233]). *A graph class $\mathcal{G}$ is minor-closed if and only if there exists a finite set $\mathcal{F}$ of graphs such that $\mathcal{G}$ is equal to the class of $\mathcal{F}$-minor-free graphs.*

We are now ready to prove our generalization of Theorem 4.2. Recall that a class of graphs is non-trivial if it does not contain all graphs.

Theorem 4.9. *Let $\mathcal{G}$ be any minor-closed graph class. Then the* Induced Minor Containment *problem is fixed parameter tractable in $|V(H)|$ on input pairs (G, H) with $G \in \mathcal{G}$ and H planar.*

Proof. Let H be a fixed planar graph with constant b_H as defined in Theorem 4.7. Let G be a graph on n vertices in a minor-closed graph class $\mathcal{G}$. From Theorem 4.8 we deduce that there exists a finite set $\mathcal{F}$ of graphs such that G is $\mathcal{F}$-minor-free. Note that $\mathcal{F}$ is non-empty, because $\mathcal{G}$ is non-trivial. By Theorem 4.6, for each $F \in \mathcal{F}$, there exists a constant c_F such that every connected F-minor-free graph of treewidth at least $c_F \cdot b_H^2$ is Γ_{b_H}-contractible or Π_{b_H}-contractible. Let $c := \min\{c_F \mid F \in \mathcal{F}\}$. We first check if $\mathrm{tw}(G) < c \cdot b_H^2$. We can do so as recognizing such graphs is fixed parameter tractable in $c \cdot b_H^2$ due to a result of Bodlaender [39].

Case 1. $\mathrm{tw}(G) < c \cdot b_H^2$. The property of having H as an induced minor is expressible in monadic second-order logic (see for example [103]). Hence, by a well-known result of Courcelle [84], we can determine in $\mathcal{O}(n)$ time if G contains H as an induced minor.

Case 2. $\mathrm{tw}(G) \geq c \cdot b_H^2$. We will show that in this case G is a yes-instance. By Theorem 4.6, we find that G is Γ_{b_H}-contractible or Π_{b_H}-contractible.

First suppose G is Γ_{b_H}-contractible. Then G has Γ_{b_H} as an induced minor. It is easy to prove that M_{b_H} has treewidth b_H. It is clear from the definition of treewidth that any supergraph of M_{b_H}, and Γ_{b_H} in particular, has treewidth at least b_H. Note that Γ_{b_H} is a planar graph. Then, by Theorem 4.7, Γ_{b_H} has H as an induced minor. Consequently, by transitivity, G has H as an induced minor.

Now suppose G is Π_{b_H}-contractible. Let $\mathcal{W}$ be a Π_{b_H}-witness structure of G. We remove all vertices in $W(s)$ from G. We then find that G has Γ_{b_H} as an induced minor and return to the previous situation. □

4.3 The H-Contractibility problem

As we mentioned at the beginning of this chapter, the presence of a dominating vertex seems to play an interesting role in the complexity classification of the H-Contractibility problem. So far, in all polynomial time solvable cases of this problem the pattern graph H has a dominating vertex, and in all NP-complete cases H does not have such a vertex. The first result of this section follows this pattern: we prove in Section 4.3.1 that $H_4^*(a_1)$-Contractibility is polynomial time solvable for all $a_1 \geq 0$. In Section 4.3.2 however we present the first class of graphs H with a dominating vertex for which H-Contractibility is NP-complete.

4.3.1 Polynomial cases with four dominating vertices

In this section, we extend Theorem 4.5 by showing that H-Contractibility is polynomial time solvable for $H = H_4^*(a_1)$ for any integer $a_1 \geq 0$.

Let H and G be graphs such that G is H-contractible. Let $\mathcal{W}$ be an H-witness structure of G. We call the subset of vertices in a witness set $W(h_i)$ that are adjacent to vertices in some other witness set $W(h_j)$ a *connector* $C_{\mathcal{W}}(h_i, h_j)$. We use the notion of connectors to simplify the witness structure of an $H_4^*(a_1)$-contractible graph. Let $y_1, \ldots, y_4$ denote the four dominating vertices of $H_4^*(a_1)$ and let $x_1, \ldots, x_{a_1}$ denote the remaining vertices of $H_4^*(a_1)$. For every $1 \leq i \leq a_1$, we define $C_{\mathcal{W}}(x_i, Y) := \bigcup_{j=1}^{4} C_{\mathcal{W}}(x_i, y_j)$, and also call such a set a connector.

The graph $H_4^*(2)$ is shown in Figure 4.2, and two copies of an $H_4^*(2)$-contractible graph G are shown in Figure 4.3. The dashed lines in the left and

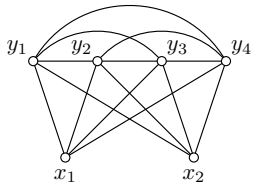

Figure 4.2: The graph $H_4^*(2)$.

the right graph indicate two different $H_4^*(2)$-witness structures $\mathcal{W}$ and $\mathcal{W}'$ of G, respectively. Exactly four vertices of the witness set $W(x_2)$ are adjacent to $W(y_1) \cup W(y_2) \cup W(y_3) \cup W(y_4)$, which means that those four vertices form the connector $C_{\mathcal{W}}(x_2, Y)$. When we consider the $H_4^*(2)$-witness structure $\mathcal{W}'$ of the right graph, we see that none of the connectors $C_{\mathcal{W}'}(x_1, Y)$ and $C_{\mathcal{W}'}(x_2, Y)$, formed by the grey vertices, contains more than two vertices.

The next lemma shows that every $H_4^*(a_1)$-contractible graph has an $H_4^*(a_1)$-witness structure $\mathcal{W}'$ where every connector of the form $C_{\mathcal{W}'}(x_i, Y)$ has size at most two.

Lemma 4.10. *Let $a_1 \geq 0$. Every $H_4^*(a_1)$-contractible graph has an $H_4^*(a_1)$-witness structure $\mathcal{W}'$ such that for every $1 \leq i \leq a_1$ one of the following two holds:*

(i) $C_{\mathcal{W}'}(x_i, Y)$ consists of one vertex, and this vertex is adjacent to all four sets $W'(y_1)$, $W'(y_2)$, $W'(y_3)$, $W'(y_4)$;

(ii) $C_{\mathcal{W}'}(x_i, Y)$ consists of two vertices, each of them adjacent to exactly two sets of $W'(y_1)$, $W'(y_2)$, $W'(y_3)$, $W'(y_4)$.

Proof. Let $\mathcal{W}$ be an $H_4^*(a_1)$-witness structure of an $H_4^*(a_1)$-contractible graph G. Below we transform $\mathcal{W}$ into a witness structure $\mathcal{W}'$ that satisfies the statement of the lemma.

From each $W(x_i)$ we move as many vertices as possible to $W(y_1) \cup \cdots \cup W(y_4)$ in a greedy way and without destroying the witness structure. This way we obtain an $H_4^*(a_1)$-witness structure $\mathcal{W}'$ of G. See Figure **??** for an example, where the $H_4^*(2)$-witness structure $\mathcal{W}'$ in the right graph is obtained from the $H_4^*(2)$-witness structure $\mathcal{W}$ on the left by performing this greedy procedure. We claim that $1 \leq |C_{\mathcal{W}'}(x_i, Y)| \leq 2$ for every $1 \leq i \leq a_1$.

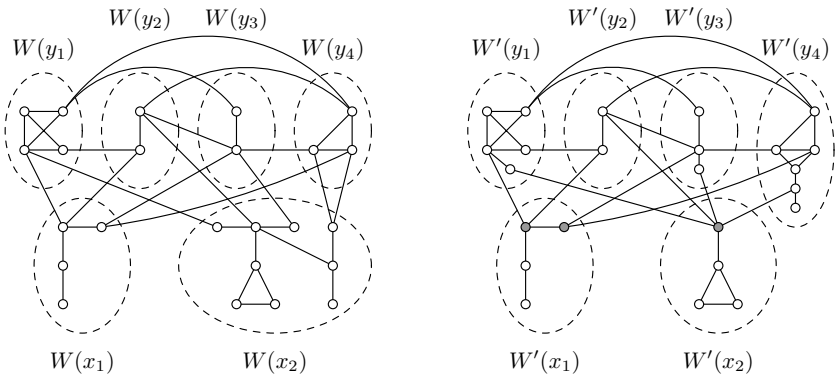

Figure 4.3: Two $H_4^*(2)$-witness structures $\mathcal{W}$ and $\mathcal{W}'$ of a graph, where $\mathcal{W}'$ is obtained from $\mathcal{W}$ by moving as many vertices as possible from $W(x_1) \cup W(x_2)$ to $W(y_1) \cup W(y_2) \cup W(y_3) \cup W(y_4)$. The grey vertices form the connectors $C_{\mathcal{W}'}(x_1, Y)$ and $C_{\mathcal{W}'}(x_2, Y)$.

Suppose, for contradiction, that $|C_{\mathcal{W}'}(x_i, Y)| \geq 3$ for some x_i. Let u_1, u_2, u_3 be three vertices in $C_{\mathcal{W}'}(x_i, Y)$. Let $L_1, \ldots, L_p$ denote the vertex sets of those components of $G[W'(x_i) \setminus \{u_1\}]$ that contain a vertex of $C_{\mathcal{W}'}(x_i, Y)$. Note that $p \geq 1$, because of the existence of u_2 and u_3. Below we prove that $p = 1$ holds.

Observe that each L_q must be adjacent to at least two "unique" witness sets from $\{W'(y_1), \ldots, W'(y_4)\}$, i.e., two witness sets that are not adjacent to $W'(x_i) \setminus L_q$, since otherwise we would have moved L_q to $W'(y_1) \cup \cdots \cup W'(y_4)$. Since u_1 is adjacent to at least one witness set, this means that $p = 1$.

The fact that $p = 1$ implies that u_1 must be adjacent to at least two "unique" witness sets from $\{W'(y_1), \ldots, W'(y_4)\}$, i.e., two witness sets that are not adjacent to $W'(x_i) \setminus \{u_1\}$; otherwise we would have moved u_1 and all components of $G[W'(x_i) \setminus \{u_1\}]$ not equal to L_1 to $W'(y_1) \cup \cdots \cup W'(y_4)$. By the same arguments, exactly the same holds for u_2 and u_3. This is not possible, as three vertices cannot be adjacent to two "unique" sets out of four. We conclude that $1 \leq |C_{\mathcal{W}'}(x_i, Y)| \leq 2$ for every $1 \leq i \leq a_1$.

Let $1 \leq i \leq a_1$. Suppose $|C_{\mathcal{W}'}(x_i, Y)| = 1$, say $C_{\mathcal{W}'}(x_i, Y) = \{p\}$. Then, by definition, p is adjacent to each of the four witness sets $W'(y_1)$,$W'(y_2)$, $W'(y_3)$, $W'(y_4)$. Suppose $|C_{\mathcal{W}'}(x_i, Y)| = 2$, say $C_{\mathcal{W}'}(x_i, Y) = \{p, q\}$. Then p is adjacent to exactly two of the sets $W'(y_1), W'(y_2), W'(y_3), W'(y_4)$, and q is adjacent to the other two sets. In all other cases we would have moved p

or q (and possibly some more vertices to keep all witness sets connected) to $W'(y_1) \cup \cdots \cup W'(y_4)$. This completes the proof of Lemma 4.10. □

We mention one more result, which can be found in the paper by Levin et al. [193], but follows directly from the polynomial time result on minors by Robertson and Seymour [232].

Lemma 4.11 ([193]). *Let G be a graph and let $Z_1, \ldots, Z_p \subseteq V(G)$ be p specified non-empty pairwise disjoint sets such that $\sum_{i=1}^{p} |Z_i| \leq k$ for some fixed integer k. The problem of deciding whether G is K_p-contractible with K_p-witness sets $U_1, \ldots U_p$ such that $Z_i \subseteq U_i$ for $i = 1, \ldots, p$ is polynomial time solvable.*

Recall that $H_4^*(0)$-Contractibility and $H_5^*(0)$-Contractibility can be solved in polynomial time by Theorem 4.5. Since $H_5^*(0) = H_4^*(1)$, this means that $H_4^*(a_1)$-Contractibility can be solved in polynomial time for $0 \leq a_1 \leq 1$. Using Lemma 4.10 and Lemma 4.11 we can prove the following result.

Theorem 4.12. *The $H_4^*(a_1)$-Contractibility problem is solvable in polynomial time for any fixed non-negative integer a_1.*

Proof. To test whether a connected graph G is $H_4^*(a_1)$-contractible, we act as follows, due to Lemma 4.10. We guess a set

$$\mathcal{S} = \{C_{\mathcal{W}'}(x_i, Y) \mid 1 \leq i \leq a_1\}$$

of connectors of size at most two. For each connector $C_{\mathcal{W}'}(x_i, Y)$ we act as follows.

If $C_{\mathcal{W}'}(x_i, Y)$ has size one, i.e., if $C_{\mathcal{W}'}(x_i, Y) = \{p\}$, then we guess four neighbors z_1, z_2, z_3, z_4 of p that are not contained in any connector of $\mathcal{S}$, and we put those vertices in sets Z_1, Z_2, Z_3, Z_4, respectively. If a connector has size two, i.e., if $C_{\mathcal{W}'}(x_i, Y) = \{p, q\}$, then we guess two neighbors z_1, z_2 of p and two neighbors z_3, z_4 of q, such that all the vertices z_1, z_2, z_3, z_4 are different and none of them belongs to any of the connectors in $\mathcal{S}$; we add vertex z_i to set Z_i for $i = 1, \ldots, 4$. We then remove the vertices of every connector in $\mathcal{S}$ from G and call the resulting graph G'.

We now check the following. First, we determine in polynomial time whether the set $Z_1 \cup Z_2 \cup Z_3 \cup Z_4$ is contained in one component D of G'. If so, we check whether D is K_4-contractible with K_4-witness sets $U_1, \ldots, U_4$ such that $Z_i \subseteq U_i$ for $i = 1, \ldots, 4$. This can be done in polynomial time due to Lemma 4.11. If not, then we guess different sets of neighbors for the same set of connectors $\mathcal{S}$ and repeat this step. Otherwise, we check whether the remaining components of G' together with the connectors $C_{\mathcal{W}'}(x_i, Y) \in \mathcal{S}$ form witness sets $W'(x_i)$ for $i = 1, \ldots, a_1$. This can be done in polynomial time; there is only one unique way to do this, because witness sets $W'(x_i)$ are not adjacent to each other. If all possible sets of neighbors of the connectors in $\mathcal{S}$ do not yield a positive answer, then we guess another set $\mathcal{S}$ of connectors and start all over. As an example, see the right graph in Figure 4.3: if we guess the three grey vertices as set $\mathcal{S}$, and all of their neighbors in $W'(y_1) \cup \ldots \cup W'(y_4)$ as the sets $Z_1, \ldots, Z_4$, then the algorithm described here would correctly decide that G is $H_4^*(2)$-contractible.

Due to Lemma 4.10 the above algorithm is correct. Since we only have to guess $\mathcal{O}(n^{2a_1})$ sets $\mathcal{S}$ with $\mathcal{O}(n^{4a_1})$ different sets of neighbors per set $\mathcal{S}$, and a_1 is fixed, it runs in polynomial time. □

4.3.2 NP-complete cases with a dominating vertex

We show the existence of a class of graphs H with a dominating vertex such that H-Contractibility is NP-complete. To do this we need the following.

Proposition 4.13. *Let H be a graph. If H-Induced Minor Containment is NP-complete, then so are $(K_1 \bowtie H)$-Contractibility and $(K_1 \bowtie H)$-Induced Minor Containment.*

Proof. Let H and G be two graphs. Write $K_1 = (\{x\}, \emptyset)$. We claim that the following three statements are equivalent.

(i) G has H as an induced minor;
(ii) $K_1 \bowtie G$ is $(K_1 \bowtie H)$-contractible;
(iii) $K_1 \bowtie G$ has $K_1 \bowtie H$ as an induced minor.

"(i) $\Rightarrow$ (ii)" Suppose G has H as an induced minor. Then, by definition, G contains an induced subgraph G' that is H-contractible. We extend an

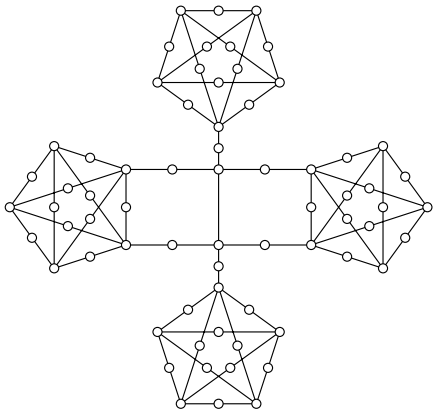

Figure 4.4: The graph $\bar{H}$.

H-witness structure $\mathcal{W}$ of G' to a $(K_1 \bowtie H)$-witness structure of $K_1 \bowtie G$ by putting x and all vertices in $V(G) \setminus V(G')$ in $W(x)$. This shows that $K_1 \bowtie G$ is $(K_1 \bowtie H)$-contractible.

"(ii) $\Rightarrow$ (iii)" Suppose $K_1 \bowtie G$ is $(K_1 \bowtie H)$-contractible. By definition, $K_1 \bowtie G$ contains $K_1 \bowtie H$ as an induced minor.

"(iii) $\Rightarrow$ (i)" Suppose $K_1 \bowtie G$ has $K_1 \bowtie H$ as an induced minor. Then $K_1 \bowtie G$ contains an induced subgraph G^* that is $K_1 \bowtie H$-contractible. Let $\mathcal{W}$ be a $(K_1 \bowtie H)$-witness structure of G^*. If $x \in V(G^*)$, then we may assume without loss of generality that $x \in W(x)$. We delete $W(x)$ and obtain an H-witness structure of the remaining subgraph of G^*. This subgraph is an induced subgraph of G. Hence, G contains H as an induced minor. □

Fellows et al. [103] showed that there exists a graph $\bar{H}$ on 68 vertices such that $\bar{H}$-Induced Minor Containment is NP-complete; this graph is depicted in Figure 4.4. Combining their result with Proposition 4.13 (applied repeatedly) leads to the following corollary.

Corollary 4.14. *For any $i \geq 1$, $(K_i \bowtie \bar{H})$-Contractibility is NP-complete.*

4.4 The (H, v)-Contractibility problem

We start with an observation. We write $K_{p,1}$ to denote the star on $p+1$ vertices with center c and leaves $b_1, \ldots, b_p$.

Observation 4.15. *The $(K_{p,1}, c)$-Contractibility problem is polynomial time solvable for all $p \geq 1$.*

Proof. Let graph $G = (V, E)$ and integer k form an instance of the $(K_{p,1}, c)$-Contractibility problem. We may without loss of generality assume that $|V| \geq k + p$. If G is $K_{p,1}$-contractible, then there exists a $K_{p,1}$-witness structure $\mathcal{W}$ of G such that $|W(b_i)| = 1$ for all $1 \leq i \leq k$. This can be seen as follows. As long as $|W(b_i)| \geq 2$ we can move vertices from $W(b_i)$ to $W(c)$ without destroying the witness structure. Our algorithm would just guess the witness sets $W(b_i)$ and check whether $V \setminus (W(b_1) \cup \cdots W(b_p))$ induces a connected subgraph. As the total number of guesses is bounded by a polynomial in p, this algorithm runs in polynomial time. □

We expect that there are relatively few pairs (H, v) for which (H, v)-Contractibility can be solved in polynomial time (under the assumption $\mathsf{P} \neq \mathsf{NP}$). This is due to the following observation and the main result in this section that shows that (H, v)-Contractibility is NP-complete whenever v is not a dominating vertex of H.

Observation 4.16. *Let H be a graph. If H-Contractibility is NP-complete, then (H, v)-Contractibility is NP-complete for every vertex $v \in V(H)$.*

Theorem 4.17. *Let H be a connected graph and let $v \in V(H)$. The (H, v)-Contractibility problem is NP-complete if v does not dominate H.*

Proof. Let H be a connected graph, and let v be a vertex of H that does not dominate H. Let $N_H(v)$ denote the neighborhood of v in H. We partition $V(H) \setminus \{v\}$ into the following three sets:

$$\begin{aligned} V_3 &:= V(H) \setminus (N_H(v) \cup \{v\}); \\ V_2 &:= \{w \in N_H(v) \mid w \text{ is not adjacent to } V_3\}; \\ V_1 &:= \{w \in N_H(v) \mid w \text{ is adjacent to } V_3\}. \end{aligned}$$

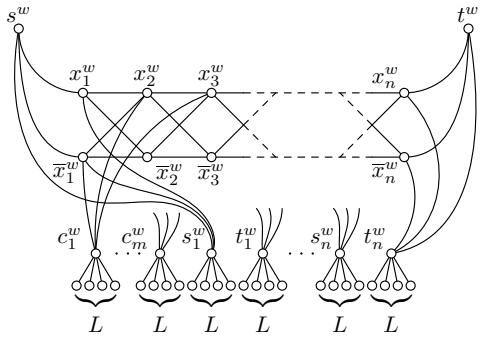

Figure 4.5: A subgraph G^w, where $c_1^w = (\overline{x}_1^w \vee x_2^w \vee x_3^w)$.

Note that neither V_1 nor V_3 is empty because H is connected and v does not dominate H; V_2 might be empty. In the top graph in Figure 4.6 a partition V_1, V_2, V_3 of the set $V_H \setminus \{v\}$ is depicted using dashed lines.

Clearly, (H, v)-Contractibility is in NP, because we can verify in polynomial time whether a given partition of the vertex set of a graph G forms an H-witness structure of G with $|W(v)| \geq k$. In order to show that (H, v)-Contractibility is NP-complete, we use a reduction from 3-Satisfiability, which is well-known to be NP-complete (cf. [127]). Let $X = \{x_1, \ldots, x_n\}$ be a set of variables and $C = \{c_1, \ldots, c_m\}$ be a set of clauses making up an instance of 3-Satisfiability. Let $\overline{X} := \{\overline{x} \mid x \in X\}$. We introduce two additional variables s and t, as well as $2n$ additional clauses $s_i := (x_i \vee \overline{x}_i \vee s)$ and $t_i := (x_i \vee \overline{x}_i \vee t)$ for $i = 1, \ldots, n$. Let $S := \{s_1, \ldots, s_n\}$ and $T := \{t_1, \ldots, t_n\}$. Note that all $2n$ clauses in $S \cup T$ are satisfied for any truth assignment. For every vertex $w \in V_1$ we create a copy X^w of the set X, and we write $X^w := \{x_1^w, \ldots, x_n^w\}$. The literals s^w, t^w and the sets $\overline{X}^w$, C^w, S^w and T^w are defined similarly for every $w \in V_1$.

We construct a graph G such that C is satisfiable if and only if G has an H-witness structure $\mathcal{W}$ with $|W(v)| \geq k$. In order to do this, we first construct a subgraph G^w of G for every $w \in V_1$ in the following way:

- every literal in $X^w \cup \overline{X}^w \cup \{s^w, t^w\}$ and every clause in $C^w \cup S^w \cup T^w$ is represented by a vertex in G^w
- we add an edge between $x \in X^w \cup \overline{X}^w \cup \{s^w, t^w\}$ and $c \in C^w \cup S^w \cup T^w$ if and only if x appears in c;
- for every $i = 1, \ldots, n-1$, we add edges $x_i^w x_{i+1}^w$, $x_i^w \overline{x}_{i+1}^w$, $\overline{x}_i^w x_{i+1}^w$, and

$\overline{x}_i^w \overline{x}_{i+1}^w$

- we add edges $s^w x_1^w$, $s^w \overline{x}_1^w$, $t^w x_n^w$, and $t^w \overline{x}_n^w$
- for every $c \in C^w \cup S^w \cup T^w$, we add L vertices whose only neighbor is c; we determine the value of L later and refer to the L vertices as the *pendant vertices*.

See Figure 4.5 for a depiction of subgraph G^w. For clarity, most of the edges between the clause vertices and the literal vertices have not been drawn. We connect these subgraphs to each other as follows. For every $w, x \in V_1$, we add an edge between s^w and s^x in G if and only if w is adjacent to x in H. Let v^* be some fixed vertex in V_1. We add an edge between $s_1^{v^*}$ and s_1^w for every $w \in V_1 \setminus \{v^*\}$. No other edges are added between vertices of two different subgraphs G^w and G^x.

We add a copy of $H[V_2 \cup V_3]$ to G as follows. Vertex $x \in V_2$ is adjacent to s^w in G if and only if x is adjacent to w in H. Vertex $x \in V_3$ is adjacent to both s^w and t^w in G if and only if x is adjacent to w in H. Finally, we connect every vertex $x \in V_2$ to $s_1^{v^*}$. See Figure 4.6 for an example.

We define $L := (2 + 2n)|V_1| + |V_2| + |V_3|$ and $k := (L + 1)(m + 2n)|V_1|$. We prove that G has an H-witness structure $\mathcal{W}$ with $|W(v)| \geq k$ if and only if C is satisfiable.

Suppose $\varphi : X \to \{T, F\}$ is a satisfying truth assignment for C. Let X_T (respectively X_F) be the variables that are set to true (respectively false) by φ. For every $w \in V_1$, we define $X_T^w := \{x_i^w \mid x_i \in X_T\}$ and $\overline{X}_T^w := \{\overline{x} \mid x \in X_T^w\}$; the sets X_F^w and $\overline{X}_F^w$ are defined similarly. We define the H-witness sets of G as follows. Let $W(w) := \{w\}$ for every $w \in V_2 \cup V_3$, and let $W(w) := \{s^w, t^w\} \cup X_F^w \cup \overline{X}_T^w$ for every $w \in V_1$. Finally, let $W(v) := V(G) \setminus (\bigcup_{w \in V_1 \cup V_2 \cup V_3} W(w))$. Note that for every $w \in V_1$ and for every $i = 1, \ldots, n$, exactly one of $x_i^w, \overline{x}_i^w$ belongs to $X_F^w \cup \overline{X}_T^w$. Hence, $G[W(w)]$ is connected for every $w \in V_1$. Since φ is a satisfying truth assignment for C, every c_i^w is adjacent to at least one vertex of $X_T^w \cup \overline{X}_F^w$ for every $w \in V_1$; by definition, this also holds for every s_i^w and t_i^w. This, together with the edges between $s_1^{v^*}$ and s_1^w for every $w \in V_1 \setminus \{v^*\}$, assures that $G[W(v)]$ is connected. So the witness set $G[W(w)]$ is connected for every $w \in V(H)$. By construction, two witness sets $W(w)$ and $W(x)$ are adjacent if and only if w and x are adjacent in H. Hence $\mathcal{W} := \{W(w) \mid w \in V(H)\}$ is an H-witness structure of G. Witness set $W(v)$ contains $n|V_1|$ literal vertices,

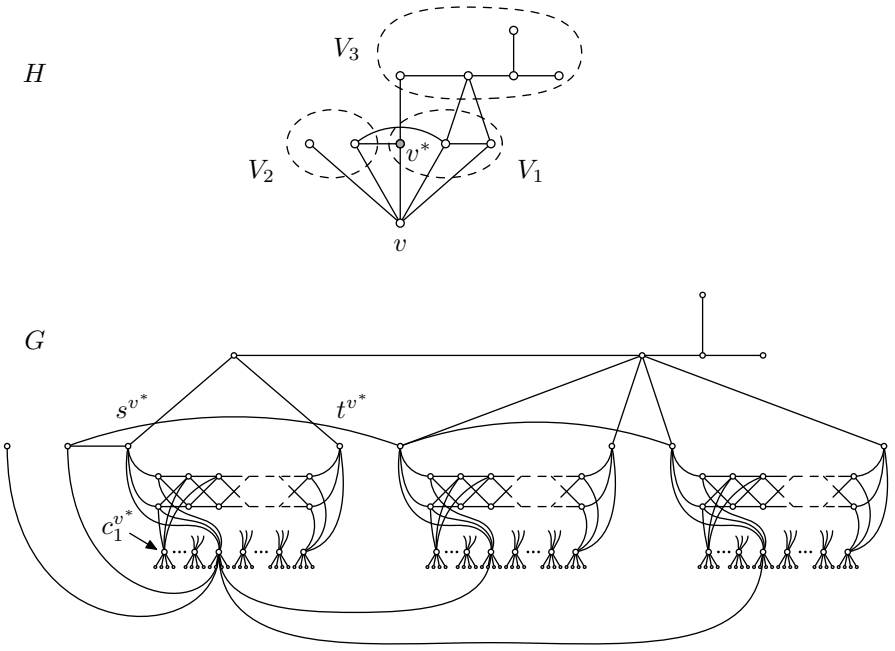

Figure 4.6: A graph H, where v^* is the grey vertex, and the corresponding graph G.

$(m + 2n)|V_1|$ clause vertices and L pendant vertices per clause vertex, i.e., $|W(v)| = (L + 1)(m + 2n)|V_1| + n|V_1| \geq k$.

Suppose G has an H-witness structure $\mathcal{W}$ with $|W(v)| \geq k$. We first show that all of the $(m + 2n)|V_1|$ clause vertices must belong to $W(v)$. Note that for every $w \in V_1$, the subgraph G^w contains $2+2n+(L+1)(m+2n)$ vertices: the vertices s^w and t^w, the $2n$ literal vertices in $X^w \cup \overline{X}^w$, the $m+2n$ clause vertices and the $L(m + 2n)$ pendant vertices. Hence we have

$$|V(G)| = (2 + 2n + (L + 1)(m + 2n))|V_1| + |V_2| + |V_3|.$$

Suppose there exists a clause vertex c that does not belong to $W(v)$. Then the L pendant vertices adjacent to c cannot belong to $W(v)$ either, as $W(v)$ is connected and the pendant vertices are only adjacent to c. This means that $W(v)$ can contain at most $|V(G)| - (L + 1) = (L + 1)(m + 2n)|V_1| - 1$ vertices, contradicting the assumption that $W(v)$ contains at least $k = (L + 1)(m + 2n)|V_1|$ vertices. So all of the $(m + 2n)|V_1|$ clause vertices, as well as all the pendant vertices, must belong to $W(v)$.

We define $W_i := \bigcup_{w \in V_i} W(w)$ for $i = 1, 2, 3$ and prove four claims.

Claim 1: $V_3 = W_3$.

The only vertices of G that are not adjacent to any of the clause vertices or pendant vertices in $W(v)$ are the vertices of V_3. As W_3 contains at least $|V_3|$ vertices, this proves Claim 1.

Claim 2: For any $w \in V_1$, both s^w and t^w belong to W_1.

Let w be a vertex in V_1, and let $w' \in V_3$ be a neighbor of w in H. Recall that both s^w and t^w are adjacent to w' in G. Suppose that s^w or t^w belongs to $W(v) \cup W_2$. By Claim 1, $w' \in W_3$. Then $W(v) \cup W_2$ and W_3 are adjacent. By construction, this is not possible. Suppose that s^w or t^w belongs to W_3. Then W_3 and $W(v)$ are adjacent, as s^w and t^w are adjacent to at least one clause vertex, which belongs to $W(v)$. This is not possible.

Claim 3: For any $w \in V_1$, at least one of each pair $x_i^w, \overline{x}_i^w$ of literal vertices belongs to $W(v)$.

Let $w \in V_1$. Suppose there exists a pair of literal vertices $x_i^w, \overline{x}_i^w$ both of which do not belong to $W(v)$. Apart from its L pendant vertices, the vertex t_i^w is only adjacent to x_i^w, $\overline{x}_i^w$ and t^w. The latter vertex belongs to W_1 due to Claim 2. Hence t_i^w and its L pendant vertices induce a component of $G[W(v)]$. Since $G[W(v)]$ contains other vertices as well, this contradicts the fact that $G[W(v)]$ is connected.

Claim 4: There exists a $w \in V_1$ for which at least one of each pair $x_i^w, \overline{x}_i^w$ of literal vertices belongs to W_1.

Let $S' := \{s^w \mid w \in V_1\}$ and $T' := \{t^w \mid w \in V_1\}$. By Claim 2, $S' \cup T' \subseteq W_1$. Suppose, for contradiction, that for every $w \in V_1$ there exists a pair $x_i^w, \overline{x}_i^w$ of literal vertices, both of which do not belong to W_1. Then for any $x \in V_1$, the witness set containing t^x does not contain any other vertex of $S' \cup T'$, as there is no path in $G[W_1]$ from t^x to any other vertex of $S' \cup T'$. But that means W_1 contains at least $|V_1|+1$ witness sets, namely $|V_1|$ witness sets containing one vertex from T', and at least one more witness set containing vertices of S'. This contradiction to the fact that W_1, by definition, contains exactly $|V_1|$ witness sets finishes the proof of Claim 4.

Let $w \in V_1$ be a vertex for which of each pair $x_i^w, \overline{x}_i^w$ of literal vertices exactly one vertex belongs to W_1 and the other vertex belongs to $W(v)$;

such a vertex w exists as a result of Claim 3 and Claim 4. Let φ be the truth assignment that sets all the literals of $X^w \cup \overline{X}^w$ that belong to $W(v)$ to true and all other literals to false. Note that the vertices in C^w form an independent set in $W(v)$. Since $G[W(v)]$ is connected, each vertex $c_i^w \in C^w$ is adjacent to at least one of the literal vertices set to true by φ. Hence φ satisfies C. □

4.5 Conclusion

The most challenging task is to find a complete computational complexity classification of both the H-Induced Minor Containment problem and the H-Contractibility problem. With regards to the second problem, all previous evidence suggested some working conjecture stating that this problem is polynomial time solvable if H contains a dominating vertex and NP-complete otherwise. However, in this chapter we presented a class of graphs H with a dominating vertex for which H-Contractibility is NP-complete. This sheds new light on the H-Contractibility problem and raises a whole range of new questions.

What is the smallest graph H that contains a dominating vertex for which H-Contractibility is NP-complete? The smallest graph known so far is the graph $K_1 \bowtie \bar{H}$, where $\bar{H}$ is the graph on 68 vertices depicted in Figure 4.4. By Observation 4.16, we deduce that $(K_1 \bowtie \bar{H}, v)$-Contractibility is NP-complete for all $v \in V(K_1 \bowtie \bar{H})$. The following question might be easier to answer: what is the smallest graph H that contains a dominating vertex v for which (H, v)-Contractibility is NP-complete? We showed that (H, v)-Contractibility is NP-complete if H is connected and v does not dominate H. We still expect a similar result for H-Contractibility, i.e., we expect that the H-Contractibility problem is NP-complete if H does not have a dominating vertex.

Lemma 4.10 plays a crucial role in the proof of Theorem 4.12 that shows that $H_4^*(a_1)$-Contractibility can be solved in polynomial time. This lemma cannot be generalized such that it holds for the $H_i^*(a_1)$-Contractibility problem for $i \geq 5$ and $a_1 \geq 2$. Hence, new techniques are required to attack the $H_i^*(a_1)$-Contractibility problem for $i \geq 5$ and $a_1 \geq 2$. Since a graph is contractible to a complete graph K_k if and only if it has K_k as a

minor, we can solve $H_i^*(1)$-Contractibility in polynomial time for every $i \geq 1$ by Theorem 4.1. Is $H_5^*(a_1)$-Contractibility solvable in polynomial time for all $a_1 \geq 2$?

We expect that the (H, v)-Contractibility problem can be solved in polynomial time for only a few target pairs (H, v). One such class of pairs might be (K_p, v), where v is an arbitrary vertex of K_p. Using similar techniques as before (i.e., simplifying the witness structure), one can easily show that (K_p, v)-Contractibility is polynomial time solvable for $p \leq 3$. Is (K_p, v)-Contractibility polynomial time solvable for all $p \geq 4$?

We finish this section with some remarks on fixing the parameter k in an instance (G, k) of the (H, v)-Contractibility problem.

Proposition 4.18. *The (P_3, p_3)-Contractibility problem is in* XP.

Proof. We first observe that any graph G that is a yes-instance of this problem has a P_3-witness structure $\mathcal{W}$ with $|W(p_1)| = 1$. This is so, as we can move all but one vertex from $W(p_1)$ to $W(p_2)$ without destroying the witness structure (see also Figure 1.1). Moreover, such a graph G contains a set $W^* \subseteq W(p_3)$ such that $|W^*| = k$ and $G[W^*]$ is connected. Hence we act as follows.

Let G be a graph. We guess a vertex v and a set V^* of size k. We put all neighbors of v in a set W_2. We check if $G[V^*]$ is connected. If so, we check for each $y \in V(G) \setminus (V^* \cup N(v) \cup \{v\})$ whether it is separated from $N(v)$ by V^* or not. If so, we put y in V^*. If not, we put y in W_2. In the end we check if $G[W_2]$ and $G[V^*]$ are connected. If so, G is a yes-instance of (P_3, p_3)-Contractibility, as $W(p_1) = \{v\}$, $W(p_2) = W_2$ and $W(p_3) = V^*$ form a P_3-witness structure of G with $|W(p_3)| \geq k$. If not, we guess another pair (v, V^*) and repeat the steps above. Since these steps can be performed in polynomial time and the total number of guesses is bounded by a polynomial in k, the result follows. □

Can we strengthen Proposition 4.18 by showing that the (P_3, p_3)-Contractibility problem also belongs to the class FPT?

Chapter 5

Computing role assignments of chordal graphs

The results presented in this chapter have appeared in preliminary form in the following paper.

[159] P. van 't Hof, D. Paulusma, and J.M.M. van Rooij. Computing role assignments of chordal graphs. In: *Proceedings of the 17th International Symposium on Fundamentals of Computation Theory (FCT 2009)*, volume 5699 of *Lecture Notes in Computer Science*, pages 193–204, Springer, 2009.

Deciding whether or not the vertex set of a graph can be partitioned into two disjoint connected sets is easy, as we explained at the very start of Chapter 3. Partitioning the vertices of a graph into two independent sets is also easy, as this is equivalent to finding a proper 2-coloring of a graph, a problem which is well-known to be solvable in linear time. In this chapter, we study yet another problem where the goal is to find a partition of the vertices of a graph into two sets. Suppose you are asked to color the vertices of a graph with two colors, such that if two vertices v_1 and v_2 receive the same color, then the set of colors assigned to the neighbors of v_1 coincides with the set of colors assigned to the neighbors of v_2. For example, if both v_1 and v_2 are colored red, and all the neighbors of v_1 are blue, then all the neighbors of v_2 must also be blue. If however v_1 and v_2 are colored the same, and v_1 has a red and a blue neighbor, then the same must hold for v_2. Clearly, a trivial

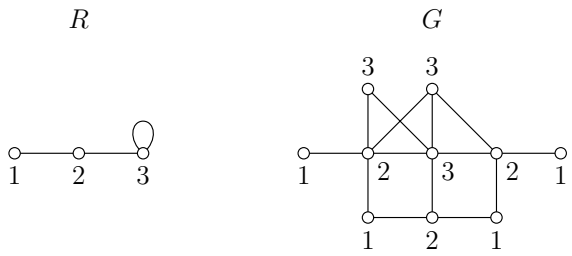

Figure 5.1: A role graph R and a graph G with an R-role assignment.

solution to this problem assigns the same color to every vertex. Interestingly, the problem becomes NP-hard if we require that both colors are used, as was proved by Roberts and Sheng [230]. However, using the structure of chordal graphs we obtain a polynomial time algorithm for solving this problem on this graph class. We also show that the problem remains NP-hard on chordal graphs when more than two colors have to be used.

5.1 Background and results

Given two graphs, say G on vertices $u_1, \ldots, u_n$ and R on vertices $1, \ldots, k$ called *roles*, an *R-role assignment* of G is a vertex mapping $r : V(G) \to V(R)$ such that the neighborhood relation is maintained, and the roles of the neighbors of each vertex u in G are exactly the neighbors of role $r(u)$ in R. In other words,

$$\text{for every } u \in V(G) : r(N_G(u)) = N_R(r(u)).$$

Here we write $r(S) = \{r(u) \mid u \in S\}$ for any subset $S \subseteq V(G)$. An R-role assignment r of G is called a *k-role assignment* of G if $|r(V(G))| = |V(R)| = k$. An equivalent definition states that r is a k-role assignment of G if r maps each vertex of G into a positive integer such that $|r(V(G))| = k$ and $r(N_G(u)) = r(N_G(u'))$ for any two vertices u and u' with $r(u) = r(u')$. See Figure 5.1 for an example; note that the R-role assignment of G in this example is also a 3-role assignment of G.

Role assignments were introduced by Everett and Borgatti [99], who called them role colorings. They originate in the theory of social behavior. The *role graph* R models roles and their relationships, and for a given

society we can ask if its individuals can be assigned roles such that relationships are preserved: each person playing a particular role has exactly the roles prescribed by the model among its neighbors. This way one investigates whether large networks of individuals can be compressed into smaller ones that still give some description of the large network. Since persons of the same social role may be related to each other, the smaller network can contain loops. In other words, given a *simple* instance graph G on n vertices does there exist a *possibly non-simple* role graph R on $k < n$ vertices in such a way that G has an R-role assignment?

From the computational complexity point of view it is interesting to know whether the existence of such an assignment can be decided quickly (in polynomial time). This leads to the following two decision problems.

R-Role Assignment
Instance: a simple graph G.
Question: does G have an R-role assignment?

k-Role Assignment
Instance: a simple graph G.
Question: does G have a k-role assignment?

Motivation for the research carried out in this chapter comes from the field of locally constrained graph homomorphisms. A *graph homomorphism* from a graph G to a graph R is a vertex mapping $r : V(G) \to V(R)$ satisfying the property that the edge $r(u)r(v)$ belongs to $E(R)$ whenever the edge uv belongs to $E(G)$. If for every $u \in V(G)$ the restriction of r to the neighborhood of u, i.e., the mapping $r_u : N_G(u) \to N_R(r(u))$, is bijective, we say that r is *locally bijective* [1, 184]. If for every $u \in V(G)$ the mapping r_u is injective, we say that r is *locally injective* [105, 106]. If for every $u \in V(G)$ the mapping r_u is surjective, r is an R-role assignment of G. In this context, r is also called a *locally surjective* homomorphism from G to R.

Locally bijective homomorphisms, also called (full) graphs coverings, have applications in distributed computing [4, 5, 38] and in constructing highly transitive regular graphs [31]. Locally injective homomorphisms, also called partial graph coverings, have applications in models of telecommunication [106] and frequency assignment [107]. Besides social network theory [99, 224, 230], locally surjective homomorphisms also have applications in distributed

computing [58].

The main computational question is whether for every graph R the problem of deciding if an input graph G has a homomorphism of given local constraint to the fixed graph R can be classified as either NP-complete or polynomial time solvable. For locally bijective and injective homomorphisms there are many partial results, see for example [106, 184] for both NP-complete and polynomial time solvable cases, but even conjecturing a classification for these two locally constrained homomorphisms is problematic. This is not the case for the locally surjective constraint and its corresponding decision problem R-Role Assignment.

First of all, Roberts and Sheng [230] show that the k-Role Assignment problem is already NP-complete for $k = 2$. Fiala and Paulusma [108] show that the k-Role Assignment problem is also NP-complete for any fixed $k \geq 3$ and classify the computational complexity of the R-Role Assignment problem. Let R be a fixed role graph without multiple edges but possibly with loops. Then the R-Role Assignment problem is solvable in polynomial time if and only if one of the following three cases holds: either R has no edge, or one of its components consists of a single vertex incident with a loop, or R is simple and bipartite and has at least one component isomorphic to an edge. In all other cases the R-Role Assignment problem is NP-complete, even for the class of bipartite graphs [108]. If the instance graphs are trees, then the R-Role Assignment problem becomes polynomial time solvable for any fixed role graph R [109].

Sheng [249] presents an elegant greedy algorithm that solves the 2-Role Assignment problem in polynomial time for chordal graphs with at most one vertex of degree 1. She also characterizes all indifference graphs that have a 2-role assignment; recall that indifference graphs are exactly the proper interval graphs [229]. In Section 5.2 we present an $\mathcal{O}(n^2)$ time algorithm for the 2-Role Assignment problem on chordal graphs, thereby settling an open problem of Sheng [249]. Contrary to the greedy algorithm in [249], which uses a perfect elimination scheme of a chordal graph with at most one pendant vertex, our algorithm works for an arbitrary chordal graph G by using a dynamic programming procedure on a clique tree of G. In Section 5.3 we prove that, for any fixed $k \geq 3$, the k-Role Assignment problem remains NP-complete on chordal graphs.

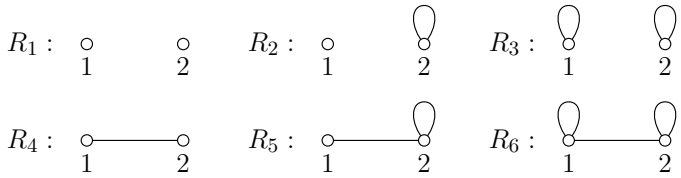

Figure 5.2: The six different role graphs on two vertices.

5.2 Computing 2-role assignments in $\mathcal{O}(n^2)$ time

In this section, we prove the following result.

Theorem 5.1. *The* 2-Role Assignment *problem can be solved in* $\mathcal{O}(n^2)$ *time for the class of chordal graphs.*

We will start by discussing the different 2-role assignments. Following the notation of Sheng [249], the six different role graphs on two vertices are:

$$\begin{aligned} R_1 &:= (\{1,2\},\emptyset);\\ R_2 &:= (\{1,2\},\{22\});\\ R_3 &:= (\{1,2\},\{11,22\});\\ R_4 &:= (\{1,2\},\{12\});\\ R_5 &:= (\{1,2\},\{12,22\});\\ R_6 &:= (\{1,2\},\{11,12,22\}). \end{aligned}$$

These six role graphs are depicted in Figure 5.2.

Let G be a chordal graph. If G contains at most one vertex, then G has no 2-role assignment. Suppose $|V(G)| \geq 2$. If G only contains isolated vertices, then G has an R_1-role assignment. If G contains at least one isolated vertex and at least one component with at least two vertices, then G has an R_2-role assignment. If G is disconnected but does not have isolated vertices, then G has an R_3-role assignment. Now assume that G is connected and has at least two vertices. If G is bipartite, then G has an R_4-role assignment. If G is not bipartite, then G has a 2-role assignment if and only if G has an R_5-role assignment or an R_6-role assignment.

We claim that we only have to check whether G has an R_5-role assignment. This is immediately clear if G has a vertex of degree 1, as such a vertex must be mapped to a role of degree 1 and R_6 does not have such a

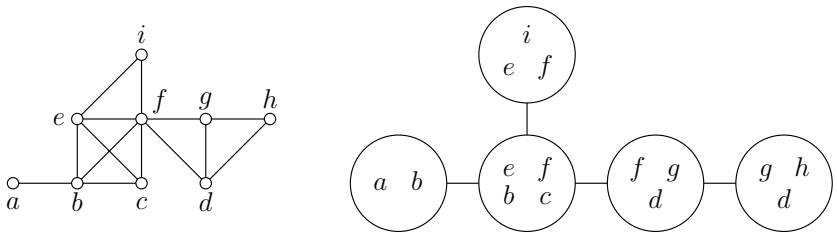

Figure 5.3: A chordal graph G (left) and a clique tree T of G.

role. If G does not have any pendant vertex, we use the following result by Sheng [249].

Theorem 5.2 ([249]). *Let G be a chordal graph with at most one vertex of degree 1 and no isolated vertices. Then G has an R_5-role assignment.*

We now present an $\mathcal{O}(n^2)$ time algorithm that solves the R_5-Role Assignment problem for chordal graphs. From the above, it is clear that this suffices to prove Theorem 5.1. Before doing this we first make some basic observations on chordal graphs.

5.2.1 On chordal graphs

Let $G = (V, E)$ be a chordal graph. Let $\mathcal{K}$ denote the set of maximal cliques of G. The *clique graph* $C(G)$ of G has as its vertex set $\mathcal{K}$, and two vertices of $C(G)$ are adjacent if and only if the intersection of the corresponding maximal cliques is non-empty. Moreover, every edge K_1K_2 of $C(G)$ is given a weight equal to $|K_1 \cap K_2|$. A tree T with vertex set $\mathcal{K}$ is a *clique tree* of G if and only if T is a maximum weight spanning tree of $C(G)$. Several alternative definitions and characterizations of clique trees have appeared in the literature (see for example [35]). See Figure 5.3 for an example of a chordal graph G and a clique tree of G.

We refer to a set $K \in \mathcal{K}$ as a *bag* of T. We define the notions *root bag*, *parent bag*, *child bag* and *leaf bag* of a clique tree similar to the notions root, parent, child and leaf of a "normal" tree. If the bag $K_r \in \mathcal{K}$ is the root bag of a clique tree T of G, then we say that T is *rooted at* K_r. A *descendant* of a bag K is a bag K^* such that K lies on the (unique) path from K^* to the root bag K_r in T. Every bag $K \neq K_r$ of a clique tree T has exactly one

parent bag K' in T. We say that a vertex $v \in K$ is *given to the parent bag* K' if $v \in K \cap K'$, i.e., if v is both in the child bag K and in the parent bag K'. We say that vertex $v \in K$ *stays behind* if $v \in K \setminus K'$, i.e., if v is in the child bag K but not in the parent bag K'. The definition of a clique tree given above immediately implies the following observation.

Observation 5.3. *Let G be a connected chordal graph with at least two maximal cliques. Let $T = (\mathcal{K}, \mathcal{E})$ be a clique tree of G rooted at K_r. At least one vertex of any bag $K \neq K_r$ of T is given to the parent bag of K and at least one vertex stays behind. Moreover, $|K| \geq 2$ for all $K \in \mathcal{K}$.*

It is well-known that a graph is chordal if and only if it has a clique tree [130]. We will make use of the following results.

Theorem 5.4 ([129])**.** *Every clique tree of a connected chordal graph has at most n bags.*

Theorem 5.5 ([256])**.** *A clique tree of a connected chordal graph can be constructed in $\mathcal{O}(n + m)$ time.*

5.2.2 An outline of our algorithm for R_5-role assignments

Our algorithm for solving the R_5-Role Assignment problem for chordal graphs takes as input a chordal graph $G = (V, E)$, and either outputs an R_5-role assignment of G, or outputs No if such a role assignment does not exist. The algorithm starts by computing a clique tree $T = (\mathcal{K}, \mathcal{E})$ of G, which can be done in $\mathcal{O}(n + m)$ time due to Theorem 5.5. The algorithm then executes two phases.

Phase 1. Decide whether or not G has an R_5-role assignment

In Phase 1, each vertex $v \in V$ is initially assigned a label $L(v) = 0$. The algorithm processes the bags of T in a "bottom-up" manner, starting with the leaf bags of T, and processing a bag K' only after all its child bags have been processed. When processing a bag, our algorithm updates the labels of the vertices of this bag in order to maintain information about the possible roles that these vertices can get in a possible R_5-role assignment of G, as well as information about the possible roles of their neighbors. To this end, it uses the labels defined in Table 5.1.

Labels of two vertices can be conflicting if they represent information on the possible roles of the vertices that cannot be combined to an R_5-role assignment. For example, no two vertices in a bag can both get label 1, because this would mean each of them must have role 1. Our algorithm first checks if there are any conflicting labels. If so, it outputs No. Otherwise, it updates the labels in order to maintain Invariant 1 below. Here, a *solution* on G is an R_5-role assignment of G. A *partial solution* on a subgraph H of G is a mapping $r' : V(H) \to \{1, 2\}$ such that no two adjacent vertices x, y of H have roles that are forbidden by R_5 (i.e., we do not have $r'(x) = r'(y) = 1$), and every vertex x in H not adjacent to a vertex in $G - H$ has neighbors with the roles required by R_5 (at least one neighbor y with role $r'(y) = 2$, and if $r'(x) = 2$, also at least one neighbor z with role $r'(z) = 1$).

Invariant 1. *Let V' be the set of vertices of G that are not in any descendant of a bag K'. Then a partial solution on $G[V' \cup K']$ can be extended to a solution on G if and only if it satisfies the constraints given by the labels of the vertices on $K \cap K'$ for every child bag K of K'.*

Suppose that at some moment bag K_r is processed. We observe that $V' \cup K' = K_r$ in Invariant 1 if $K' = K_r$. Hence, our algorithm ensures that a partial solution on $G[V' \cup K'] = G[K_r]$ can be extended to a solution on G if and only if it satisfies the constraints given by the labels of the vertices on $K \cap K_r$ for every child bag K of K_r. Our algorithm will now decide if such a partial solution on $G[K_r]$ exists. If so, it finds one and goes to Phase 2. If not, it outputs No.

Phase 2. Produce an R_5-role assignment of G

The partial solution on $G[K_r]$ found at the end of Phase 1 is propagated to a solution of G in a "top-down" manner, processing a bag K only after its parent bag has been processed. Every time a bag is processed, roles are assigned in a greedy way such that the constraints imposed by the labels of vertices it has in common with its child bags are satisfied.

5.2.3 Phase 1 in detail

Table 5.1 shows what labels a vertex v in a bag K can have. We observe that Phase 2 is only executed if G is indeed R_5-role assignable. We implicitly

$L(v) = 0$	initial label of every vertex
$L(v) = 1^*$	v belongs to a set $J \subseteq K$ of at least two vertices that are all labeled 1^* because they are given to K from the same child bag in which a vertex with label 2_1 is left behind; exactly one vertex in J must get role 1 (and hence all others must get role 2)
$L(v) = 1$	v must get role 1 in Phase 2
$L(v) = 2$	v must get role 2 in Phase 2; it is ensured that v gets a neighbor with role 1 and a neighbor with role 2
$L(v) = 2_1$	v must get role 2 in Phase 2; it is ensured that v gets a neighbor with role 2, but we must still make sure that v gets a neighbor with role 1
$L(v) = 2_2$	v must get role 2 in Phase 2; it is ensured that v gets a neighbor with role 1, but we must still make sure that v gets a neighbor with role 2
$L(v) = 1\|2$	v may get either role 1 or 2 in Phase 2; in the latter case it is ensured that v gets a neighbor with role 1 and a neighbor with role 2
$L(v) = 1\|2_1$	v may get either role 1 or 2 in Phase 2; in the latter case it is ensured that v gets a neighbor with role 2, but we must still make sure that v gets a neighbor with role 1
$L(v) = 1\|2_2$	v may get either role 1 or 2 in Phase 2; in the latter case it is ensured that v gets a neighbor with role 1, but we must still make sure that v gets a neighbor with role 2

Table 5.1: The different labels a vertex v can have.

assume this in Table 5.1 and in the remainder of this section, whenever we write that some vertex gets some role in Phase 2.

Initially, every vertex has label 0, but the label of a vertex can change several times: the arrows in Figure 5.4 represent all possible transitions between two labels. This figure will be clarified in detail later on. For now, we only note that no arrows point downwards in Figure 5.4. This corresponds to the fact that labels in a higher level contain more information than labels in a lower level. For example, if a vertex v in bag K has a label 2_2 and one of its neighbors in K gets label 2, then we change the label of v into 2 before processing the parent bag of K. After all, label 2 contains more information than label 2_2, as label 2 contains the information that at least one neighbor of v will get role 2 in Phase 2.

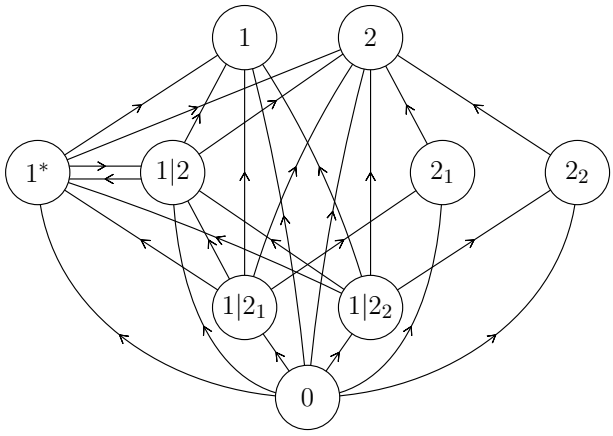

Figure 5.4: All possible labels and all possible transitions between them.

Recall that a bag is processed only if all its child bags have already been processed. Let K be the bag that is currently being processed. First assume $K \neq K_r$, and let K' be the parent bag of K. By Observation **??**, at least one vertex in K stays behind, and at least one vertex is given to K'. We make sure that when our algorithm moves to K' each vertex $u \in K \cap K'$ has $L(u) \neq 0$.

Phase 1a. Deal with leaf bags

Suppose K is a leaf bag of T. Let v be a vertex that stays behind.

Suppose $|K| = 2$. Then v has degree 1 in G. Let x be the other vertex of K. By Observation **??**, we find that x is given to K'. Because v has degree 1 in G, we must set $L(v) := 1$ and $L(x) := 2_2$, as v must get role 1, x must get role 2, and we must ensure that at least one other neighbor of x gets role 2.

Suppose $|K| \geq 3$. We assign label $1|2$ to every vertex in K. We do so because of the following. If in Phase 2 all vertices in $K \cap K'$ receive role 2, then we assign role 1 to v and role 2 to all other vertices of $K \setminus K'$. If one of the vertices in $K \cap K'$ receives role 1, then we assign role 2 to v and to all other vertices of $K \setminus K'$.

Phase 1b. Deal with non-leaf bags that are not the root bag

Suppose K is not a leaf bag of T. Recall that we process K only after each of its child bags has been processed. Hence, every vertex in K that is given

to K from a child bag does not have label 0. However, vertices of K might belong to different child bags, and consequently, they might have received different labels. We show how to combine these multiple labels into a single label such that Invariant 1 is maintained.

Case 1: K contains a vertex v that received label 1 in at least one of the child bags of K.

We output No in the following cases:

- v has received label $2, 2_1$ or 2_2 in another child bag of K;
- there is a vertex $w \in K \setminus \{v\}$ that received label 1 in a child bag of K;
- there is a set $J \subseteq K$ of vertices that received label 1^* in a child bag of K to which v does not belong.

If the above three cases do not occur we do as follows. We set $L(v) := 1$ and observe that all other vertices in K must get role 2 in Phase 2.

If $|K| \geq 3$, we set $L(w) := 2$ for every $w \in K \setminus \{v\}$; as at least two vertices get role 2, all vertices in $K \setminus \{v\}$ will have both a neighbor with role 1 (namely v) and a neighbor with role 2. We proceed to the next bag.

If $|K| = 2$, then for the only vertex $w \in K \setminus \{v\}$, we set $L(w) := 2_2$ if either the label of w in K is 0, or else belongs to $\{1|2_2, 2_2\}$ in every child bag of K that contains w. This is because in both cases we must still ensure that w gets a neighbor with role 2 in Phase 2. In the other case w has received a label from $\{1^*, 1|2, 1|2_1, 2_1\}$ in at least one child bag of K. Then we set $L(w) := 2$. The reason for this is that w is ensured to get a neighbor with role 2 in Phase 2. If w received label $1|2, 1|2_1$ or 2_1 in a child bag of K, then this follows directly from the definition of its label. If w received label 1^* in a child bag of K, it also follows from the definition of its label: a vertex with label 1^* must have a neighbor with label 2_1 and this neighbor will get role 2 in Phase 2. We proceed to the next bag.

Case 2: Case 1 does not occur but K contains a vertex that received label 1^* in at least one of the child bags of K.

For some $p \geq 1$, let K contain sets $V_1, \ldots, V_p$ of vertices that have label 1^*, each originating from a different child bag. Note that a vertex can be in more than one set V_i.

For $i = 1, \ldots, p$, let V_i' be the set of vertices of V_i that did not receive label $2, 2_1$ or 2_2 in any child bag of K. If $V_i' = \emptyset$ for some i we output No, since in that case there is no vertex in V_i that will get the required role 1 in Phase 2.

Suppose $V_i' \neq \emptyset$ for all $1 \leq i \leq p$. Note that exactly one vertex of each V_i' must get role 1 in Phase 2. As K may not contain two vertices that get role 1, such a vertex must be in $V^* := \bigcap_{i=1}^{p} V_i'$. If $|V^*| = 0$, this means that we must output No.

If $|V^*| = 1$, then the only vertex in V^* must receive role 1 in Phase 2 in order to satisfy the constraint imposed by label 1^* with respect to every V_i. Let v be this vertex. We set $L(v) := 1$. Moreover, we set $L(w) := 2$ for every $w \in K \setminus \{v\}$ for the following reasons. Firstly, none of the vertices in $K \setminus \{v\}$ received label 1 in any of the child bags of K, since we assumed that Case 1 does not occur. Secondly, the constraint imposed by label 1^* with respect to every V_i will be satisfied by v. Thirdly, every vertex in $K \setminus \{v\}$ has a neighbor that will get role 1 in Phase 2, namely v, and a neighbor that will get role 2. The latter is immediately clear if $|K| \geq 3$. We show that $|K| = 2$ does not occur. Let w be the only vertex in $K \setminus \{v\}$. Then w must have received label 1^* in a child bag of K as well, because by definition of this label such vertices come in groups of size at least 2. But then K is a subset of that child bag, which means that none of the vertices of that child bag stays behind, contradicting Observation **??**. We proceed to the next bag.

If $|V^*| \geq 2$, we do as follows. We set $L(w) := 2$ for all $w \in K \setminus V^*$ for the same three reasons as in the case $|V^*| = 1$; the only difference is that we do not know which vertex in V^* will get the required role 1. For determining the label of the vertices in V^* we distinguish the following two situations.

If $V^* \subseteq (K \cap K')$ we set $L(u) := 1^*$ for every $u \in V^*$. If at least one vertex $v \in V^*$ stays behind then we set $L(v) = 1|2$ for every vertex in V^*. After all, if a vertex $x \in V^* \cap K'$ receives role 1 in Phase 2, then all neighbors of x (including v) must receive role 2. If all vertices in $V^* \cap K'$ receive role 2 (or if $V^* \cap K' = \emptyset$), then we can give the required role 1 to v. In both situations we proceed to the next bag.

From now on we assume that Case 1 and Case 2 do not hold for bag K, so K does not contain a vertex that received label 1 or 1^* in one of the child bags of K. It is still possible, however, that a vertex $v \in K$ received labels

	2	2_1	2_2	$1\|2$	$1\|2_1$	$1\|2_2$
2	2	2	2	2	2	2
2_1	2	2_1	2	2	2_1	2
2_2	2	2	2_2	2	2	2_2
$1\|2$	2	2	2	$1\|2$	$1\|2$	$1\|2$
$1\|2_1$	2	2_1	2	$1\|2$	$1\|2_1$	$1\|2$
$1\|2_2$	2	2	2_2	$1\|2$	$1\|2$	$1\|2_2$

Table 5.2: Combining two labels from different child bags.

in two or more different child bags. In case v received labels from more than two different child bags, we first combine two labels into a new label, then combine this new label with the next label, and continue until a single label remains. Below we give the rules on how to combine the labels that v received in two different child bags K_1 and K_2 of K into a single label.

Suppose $v \in K$ has label 2_1 in K_1 and label 2_2 in K_2. Label 2_1 means that v is ensured to have a neighbor that will receive role 2 in Phase 2. Label 2_2 means that v is ensured to have a neighbor that will receive role 1 in Phase 2. Hence we set $L(v) := 2$. Suppose $v \in K$ has label 2_1 in K_1 and label $1|2_2$ in K_2. Then v cannot get role 1 in Phase 2. Hence we set $L(v) := 2$.

Arguments like the above, directly following from the definitions of the labels, can be used for all other combinations. See Table 5.2 for an overview of how the possible combinations of two labels are combined into a new label. From now on suppose that every vertex v of K has exactly one label $L(v)$. We write $L(K) := \{L(v) \mid v \in K\}$ and distinguish several cases.

Case 3: $\{2_2\} \subseteq L(K) \subseteq \{0, 1|2, 1|2_1, 1|2_2, 2, 2_1, 2_2\}$.

Let $v \in K$ have $L(v) = 2_2$. If K contains another vertex with label 2, 2_1 or 2_2, then any $v' \in K$ with $L(v') = 2_2$ (including vertex v) has its required role 2 neighbor and we set $L(v') := 2$ for all such v'. Furthermore, if $|K| \geq 3$, then at least one vertex in $K \setminus \{v\}$ will get role 2 and we also set $L(v') := 2$ for all $v' \in K$ with $L(v') = 2_2$. In both cases we arrive at Case 5, which we discuss later on.

The case where $|K| = 2$ remains; let x be the other vertex in K. If $L(x) = 1|2_2$, then we replace it by $L(x) := 1|2$ since neighbor v will receive role 2 in Phase 2. This implies that we may assume that $L(x) \in \{0, 1|2, 1|2_1\}$.

Now, either x stays behind, or x is given to K'.

If x stays behind, then v is given to K' by Observation **??**. The label of x can only be influenced by v. If $L(x) = 1|2$, then x can function as the neighbor with role 2 that v needs, so we set $L(v) := 2$ and $L(x) := 2$. If $L(x) \in \{0, 1|2_1\}$, then the fact that none of the neighbors of x will receive role 1 (otherwise x would not have label 0 or $1|2_1$) means we must set $L(x) := 1$, and consequently, leave $L(v) = 2_2$ unaltered. We proceed to the next bag.

If x is given to K', then v stays behind while it still needs a neighbor with role 2, which can only be x. Hence, we apply the following relabeling to x that forces x to get role 2: $0 \to 2_1$, $1|2 \to 2$, $1|2_1 \to 2_1$. We proceed to the next bag.

Case 4: $\{2_1\} \subseteq L(K) \subseteq \{0, 1|2, 1|2_1, 1|2_2, 2, 2_1\}$.

Let $v \in K$ have $L(v) = 2_1$. Similar to the previous case, we replace any label $1|2_2$ in K by $1|2$. Hence we may assume that $L(x) \in \{0, 1|2, 1|2_1, 2, 2_1\}$ for every $x \in K \setminus \{v\}$.

Suppose there exists a vertex $x \in K \setminus \{v\}$ with $L(x) \in \{0, 1|2, 1|2_1\}$ that stays behind. Due to this vertex we change the label of every vertex $v' \in K \setminus \{x\}$, and in particular of every $v' \in K \cap K'$, as follows. If $L(v') \in \{0, 1|2, 1|2_1\}$, then we set $L(v') := 1|2$, and if $L(v') \in \{2, 2_1\}$, then we set $L(v') := 2$. This is so, since if none of the vertices in $K \cap K'$ receives role 1 in Phase 2, then Phase 2 assigns role 1 to x; otherwise x gets role 2. The latter is fine since v will also receive role 2. We proceed to the next bag.

Suppose $L(x) = 2$ for every vertex x that stays behind. For every vertex $v' \in K$ with $L(v') = 0$, we set $L(v') := 1|2_1$. We may do this because v' either gets role 1, or gets role 2 in which case it needs at least one neighbor that gets role 1 in Phase 2. We leave all other labels unaltered. We proceed to the next bag.

In the remaining case, at least one vertex with label 2_1 stays behind; without loss of generality, assume that v is such a vertex. If $K \cap K'$ does not contain a vertex x with $L(x) \in \{0, 1|2, 1|2_1\}$, then v will never have a neighbor with role 1 and the algorithm outputs NO. If $K \cap K'$ contains exactly one vertex x with $L(x) \in \{0, 1|2, 1|2_1\}$, this is the only vertex that can be the role 1 neighbor of v; hence we set $L(x) := 1$. If $K \cap K'$ contains multiple vertices with a label from $\{0, 1|2, 1|2_1\}$, then we set the label of each of them to 1^*. Furthermore, since there will now be a vertex with role 1 in

K, we replace any label 2_1, $1|2_1$ or $1|2$ in K by 2, before we proceed to the next bag.

Case 5: $\{2\} \subseteq L(K) \subseteq \{0, 1|2, 1|2_1, 1|2_2, 2\}$.

Since v will receive role 2, we change the label of every vertex $x \in K$ with $L(x) = 0$ or $L(x) = 1|2_2$ into $L(x) := 1|2_1$ or $L(x) := 1|2$, respectively. Hence we may assume that $L(x) \in \{1|2, 1|2_1, 2\}$ for every vertex $x \in K$. Suppose x is a vertex of K that stays behind with $L(x) \in \{1|2, 1|2_1\}$. Then, just as before, x allows us to change any label $1|2_1$ in $K \cap K'$ to label $1|2$; x will be the neighbor with role 1 if necessary. Otherwise, if $L(x) = 2$ for all $x \in K \setminus K'$, then we leave all labels in K unaltered. We proceed to the next bag.

Case 6: $L(v) \in \{0, 1|2, 1|2_1, 1|2_2\}$ *for every* $v \in K$.

By Observation **??** at least one vertex stays behind and we distinguish four cases. After each case we proceed to the next bag.

Case 6a: there exists an $x \in K$ *with* $L(x) = 1|2$ *that stays behind.*
If $|K| \geq 3$, then x allows us to set $L(v) := 1|2$ for all $v \in K$. This is true, because in Phase 2 each vertex in K will get a neighbor with role 2 (as $|K| \geq 3$), and if no vertex of $K \cap K'$ gets role 1 in Phase 2, we give role 1 to x.

Suppose $|K| = 2$. Let $K = \{x, y\}$ where y is given to K'. If y gets role 1 in Phase 2, then x will get role 2. If y gets role 2, then it already has a neighbor in K' (as $K' \setminus K \neq \emptyset$ because K' is a maximal clique) with some role and we set x to have the other. Hence we set $L(y) := 1|2$.

Case 6b: there exists an $x \in K$ *with* $L(x) = 1|2_1$ *that stays behind.*
Suppose $|K| \geq 3$. For the same reasons as in Case 6a, we assign label $1|2$ to every vertex in $K \cap K'$. Suppose $|K| = 2$. Let $K = \{x, y\}$ where y is given to K'. Notice that if y receives role 1 in Phase 2, then we can assign role 2 to x. If y receives role 2 in Phase 2, then x must get role 1, since otherwise it has no role 1 neighbor. As a result, we apply the following replacements for the label of y: $0 \to 1|2_2$, $1|2 \to 1|2$, $1|2_1 \to 1|2$, $1|2_2 \to 1|2_2$.

Case 6c: there exists an x *with* $L(x) = 1|2_2$ *that stays behind.*
If $|K| \geq 3$, then at least one vertex in $K \setminus \{x\}$ gets role 2 in Phase 2. Hence we change the label of x into $1|2$ and return to Case 6a. We may therefore

assume that $|K| = 2$. Let $K = \{x, y\}$ where y is given to K'. In this case, y cannot get role 1 because then x must get role 2. This is not possible, as then x does not have a neighbor with role 2. We maintain Invariant 1 as follows. If $L(y) \in \{1|2, 1|2_1\}$, then we set $L(y) := 2$ and $L(x) := 1$. If $L(y) = 1|2_2$, then we set $L(y) := 2$ and $L(x) := 2$. If $L(y) = 0$, we also set $L(y) := 2$, as a neighbor of y in $K' \setminus K$ gets some role in Phase 2, and we give x the other role.

Case 6d: there exists an x with $L(x) = 0$ that stays behind.
If $|K| \geq 3$, then at least one vertex in $K \setminus \{x\}$ gets role 2. Hence we may change $L(x)$ into $1|2_1$ and return to Case 6b. Thus we may assume that $|K| = 2$. Let $K = \{x, y\}$ where y is given to K'. As x has label 0 and stays behind, x must have degree 1 in G. Then x must get role 1, and y must get role 2. Hence we set $L(x) := 1$ and we apply the following replacement rules for the label of y: $1|2 \to 2$, $1|2_1 \to 2$, $1|2_2 \to 2_2$. Note that $L(y) \neq 0$ due to the assumption that K is not a leaf bag of T.

Phase 1c. Deal with the root bag

The root bag K_r is the last bag of T to be processed in Phase 1. Since the root bag is the only bag of T that does not have a parent bag, the case analysis for K_r slightly differs from the case analysis for other bags of T.

Case 1′: K_r contains a vertex v that received label 1 in at least one of the child bags of K_r.

The algorithm acts like it does in Case 1 of Phase 1b, with the following exceptions. Since the root bag is the last bag that is being processed in Phase 1, the algorithm cannot proceed to the next bag as it does in Case 1: this time, the algorithm proceeds to Phase 2 instead. The other exception is when $|K_r| = 2$ and the only vertex $w \in K_r \setminus \{v\}$ has label 0 or it has received a label from $\{1|2_2, 2_2\}$ in every child bag of K_r. Instead of setting $L(w) := 2_2$ like in Case 1, we output No in this case. This is because w will not get a role 2 neighbor in Phase 2, which means G is not R_5-role assignable.

Case 2′: Case 1′ does not occur but K_r contains a vertex that received label 1^* in at least one of the child bags of K_r.

The algorithm acts like it does in Case 2 of Phase 1b, with the following exceptions. Just like in Case 1′, the algorithm proceeds to Phase 2 instead

of to the next bag. The other exception is when $|V^*| \geq 2$. We first set $L(w) := 2$ for all $w \in K_r \setminus V^*$ for the same reasons as in Case 2 of Phase 1b. However, we then arbitrarily choose a vertex $v \in V^*$, and set $L(v) := 1$. We assign label 2 to every other vertex of V^*, and proceed to Phase 2.

Suppose Case 1′ and Case 2′ do not hold for bag K_r. Since K_r might have more than one child bag, vertices in K_r might have received labels in two or more different child bags. For each vertex $v \in K_r$ we combine these multiple labels into single label using the procedure described in Phase 1b: Table 5.2 shows how to combine the labels v received in two different child bags K_1 and K_2 of K_r. Hence, we deduce that every vertex of K_r has exactly one label and this label belongs to $\{0, 1|2, 1|2_1, 1|2_2, 2, 2_1, 2_2\}$. We distinguish two cases.

Case 3′: K_r contains a vertex v with $L(v) \in \{1|2, 1|2_1, 1|2_2\}$.

If $|K_r| \geq 3$, then we set $L(v) := 1$ and $L(x) := 2$ for every $x \in K_r \setminus \{v\}$. Since K_r contains at least three vertices, only one of which receives role 1 in Phase 2, every vertex in K_r has a role 2 neighbor, while v is the required role 1 neighbor for every vertex in $K_r \setminus \{v\}$.

Suppose $|K_r| = 2$, and let w be the only other vertex of K_r. For the same reasons as in the case $|K_r| \geq 3$, we set $L(v) := 1$ and $L(w) := 2$ if $L(w) \in \{1|2, 1|2_1, 2_1\}$. We may not do this if $L(w) \in \{0, 1|2_2, 2_2\}$, as in that case w will not have a role 2 neighbor. Suppose $L(w) \in \{0, 1|2_2, 2_2\}$. If $L(v) \in \{1|2, 1|2_1\}$ and $L(w) \in \{0, 1|2_2\}$ then we set $L(v) := 2$ and $L(w) := 1$. If $L(v) = 1|2$ and $L(w) = 2_2$, or if $L(v) = 1|2_2$ and $L(w) \in \{1|2_2, 2_2\}$, then we set $L(v) := 2$ and $L(w) := 2$. In both cases we proceed to Phase 2 afterwards. In the remaining two cases, i.e., when $L(v) = 1|2_1$ and $L(w) = 2_2$, or when $L(v) = 1|2_2$ and $L(w) = 0$, we output No.

Case 4′: $L(v) \in \{0, 2, 2_1, 2_2\}$ for every $v \in K_r$.

Let $V^* \subseteq K_r$ be the set of vertices that have label 0. Suppose $V^* = \emptyset$. If there exists a vertex $x \in K_r$ with $L(x) = 2_1$, then we output No. After all, every vertex in K_r will receive role 2 in Phase 2, which means that x will not have its required role 1 neighbor. If $L(v) \in \{2, 2_2\}$ for every $v \in K_r$, then we set $L(v) := 2$ for every $v \in K_r$ and proceed to Phase 2.

Suppose $V^* \neq \emptyset$. If $|K| \geq 3$, then every vertex $v \in V^*$ will receive a neighbor with role 2: we can replace the label 0 for all $v \in V^*$ with the label

$1|2_1$ and go to Case 3′. Suppose $|K| = 2$. If $|V^*| = 2$, then G is a graph on two vertices and we output No. Let v be the only vertex in V^*. Then v must be of degree 1 in G, so we set $L(v) := 1$. Now, if the other vertex $w \in K$ has label 2 or 2_1, we set $L(w) := 2$ and proceed to Phase 2. If $L(w) = 2_2$, then we output No.

5.2.4 Proof of correctness and running time analysis

Theorem 5.6. *The R_5-Role Assignment problem can be solved in $\mathcal{O}(n^2)$ time for the class of chordal graphs.*

Proof. Let $G = (V, E)$ be a connected chordal graph. We apply the algorithm that we described in this section. After computing a clique tree $T = (\mathcal{K}, \mathcal{E})$ of G, we set $L(u) := 0$ for every $u \in V$ and start with Phase 1. Because we maintain Invariant 1 every time we process a bag, Phase 1 and consequently Phase 2 are correct. Hence, our algorithm is correct.

We now perform a running time analysis. Computing a clique tree T of G can be done in $\mathcal{O}(n+m)$ time due to Theorem 5.5. The Phase 1 operation "compile a list of labels that v received in the child bags of a bag K to which v belongs" costs $\mathcal{O}(n)$ time for each $v \in V$, since every vertex v is in at most $\mathcal{O}(n)$ bags of T due to Theorem 5.4. Hence, all these operations together cost $\mathcal{O}(n^2)$ time. We observe that, after compiling a list of labels for a vertex v in a bag K, determining the new label of v can be done in constant time, as this list of labels has size at most 9.

Consider a bag K of T in which each vertex has exactly one label (possibly after combining labels that this vertex has in different child bags of K). Determining $L(K)$ costs $\mathcal{O}(n)$ time. We must do this in order to know how to determine the new label of each vertex in K. From the description of the algorithm it then follows that the process of assigning a new label to every vertex in K costs $\mathcal{O}(n)$ time as well. Hence, since there are at most $\mathcal{O}(n)$ bags, this part of the algorithm also costs $\mathcal{O}(n^2)$ time. We conclude that Phase 1 runs in $\mathcal{O}(n^2)$ time.

Due to our greedy approach in Phase 2, this phase can be executed in $\mathcal{O}(n^2)$ time. We conclude that the overall running time of our algorithm is $\mathcal{O}(n^2)$. This completes the proof of Theorem 5.6. □

5.2.5 A remark regarding R_6-role assignments

In our $\mathcal{O}(n^2)$ time algorithm that solves the 2-Role Assignment problem for chordal graphs we do not have to check if the input graph has an R_6-role assignment as a result of Theorem 5.2. This is rather "fortunate" as the R_6-Role Assignment problem turns out to be NP-complete even when restricted to split graphs, a subclass of chordal graphs. We show this by using a reduction from the Hypergraph 2-Colorability problem restricted to the class of *non-trivial* hypergraphs, where a hypergraph H is called *non-trivial* if Q contains at least three vertices and Q is a member of $\mathcal{S}$. The Hypergraph 2-Colorability problem clearly remains NP-complete under this restriction.

Proposition 5.7. *The R_6-Role Assignment problem is NP-complete for the class of split graphs.*

Proof. Let $(Q, \mathcal{S})$ be a non-trivial hypergraph. In its incidence graph I we add an edge between every pair of vertices in Q. This results in a split graph G. We claim that $(Q, \mathcal{S})$ has a 2-coloring if and only if G has an R_6-role assignment.

Suppose $(Q, \mathcal{S})$ has a 2-coloring (Q_1, Q_2). Since $|Q| \geq 3$, we may without loss of generality assume that $|Q_2| \geq 2$. We give each $q \in Q_1$ role 1 and each $q \in Q_2$ role 2. We assign role 1 to each $S \in \mathcal{S}$. Because (Q_1, Q_2) is a 2-coloring, each vertex in $\mathcal{S}$ has a neighbor with role 1 and a neighbor with role 2 in G. Because Q is a clique in G, each vertex in Q_2 has a neighbor with role 1 and a neighbor with role 2. For the same reason, each vertex in Q_1 has a neighbor with role 2. Since $(Q, \mathcal{S})$ is non-trivial, $Q \in \mathcal{S}$. This guarantees that also in case $|Q_1| = 1$, each vertex in Q_1 has a neighbor with role 1. We conclude that G has an R_6-role assignment.

Suppose G has an R_6-role assignment. Then every vertex in $\mathcal{S}$ has a neighbor with role 1 and a neighbor with role 2. By construction, these neighbors are in Q. This immediately gives a 2-coloring of $(Q, \mathcal{S})$. □

5.3 Complexity of k-Role Assignment for $k \geq 3$

It is known that the k-Role Assignment problem is NP-complete for any fixed $k \geq 2$ [108]. In this section, we show that the k-Role Assignment

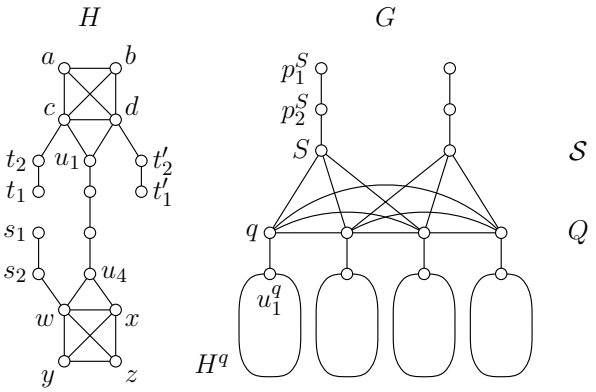

Figure 5.5: The graph H and the graph G when $k = 4$.

problem for chordal graphs is NP-complete for $k \geq 3$. We use a reduction from the Hypergraph 2-Colorability problem. Our NP-completeness proof is more involved than the one for the general case in [108] as the graph constructed there (also from an instance of Hypergraph 2-Colorability) is not chordal.

Theorem 5.8. *For $k \geq 3$, the k-Role Assignment problem is NP-complete for the class of chordal graphs.*

Proof. Let $k \geq 3$. We use a reduction from Hypergraph 2-Colorability. Let $(Q, \mathcal{S})$ be a non-trivial hypergraph with incidence graph I.

We modify I as follows. Firstly, we add an edge between any two vertices in Q, so Q becomes a clique. Secondly, for each $S \in \mathcal{S}$ we take a path $P^S = p_1^S \cdots p_{k-2}^S$ and connect it to S by the edge $p_{k-2}^S S$, so these new paths P^S are pendant paths in the resulting graph. Thirdly, we add a copy H^q of a new graph H for each $q \in Q$. Before we explain how to do this, we first define H. Start with a path $u_1 u_2 \cdots u_{2k-4}$. Then take a complete graph on four vertices a, b, c, d, and a complete graph on four vertices w, x, y, z. Add the edges $cu_1, du_1, u_{2k-4}w, u_{2k-4}x$. We then take three paths $S = s_1 \cdots s_{k-2}$, $T = t_1 \cdots t_{k-2}$ and $T' = t'_1 \cdots t'_{k-2}$, and we add the edges $s_{k-2}w, ct_{k-2}, dt'_{k-2}$. This finishes the construction of H. We connect a copy H^q to q via the edge qu_1^q, where u_1^q is the copy of the vertex u_1. We call the resulting graph G; notice that this is a connected chordal graph. See Figure 5.5 for an example.

Suppose G has a k-role assignment r, and let R be a graph on k vertices such that G has an R-role assignment. We show that we must have $R = R^*$,

where R^* denotes the path $r_1 \cdots r_k$ on k vertices with a loop in vertices r_{k-1} and r_k. To see this, consider a copy H^q of H in G; we show that we can assign roles to the vertices of H^q in only one way. For convenience, we denote the vertices of H^q without the superscript q.

Let P be an induced path in G on at most k vertices that starts in a vertex of degree 1 in G. The k-role assignment r of G must map P to an induced path on $|V(P)|$ vertices in R; otherwise, if r maps P to an induced path on less than $|V(P)|$ vertices in R, then $|r(V(G))| = |r(V(P))| < |V(P)| \leq k$, contradicting the assumption that r is a k-role assignment. Hence, we may write $r(t_i) = i$ for $i = 1, \ldots, k-2$ and $r(c) = k-1$. This implies that a vertex with role 1 only has vertices with role 2 in its neighborhood and a vertex with role i for $2 \leq i \leq k-2$ only has vertices with role $i-1$ and role $i+1$ as neighbors. Then a vertex with role k can only be adjacent to vertices with role $k-1$ or role k. Hence c must have a neighbor with role k.

Suppose $r(d) = k$. Then $r(t'_{k-2}) \in \{k-1, k\}$ and this eventually leads to $r(t'_1) \geq 2$ without a neighbor of role $r(t'_1) - 1$ for t'_1. This is not possible. Hence $r(d) \neq k$. This means that $k \in r(\{a, b, u_1\})$. Since a, b, u_1 are neighbors of d as well and a vertex with role k can only have neighbors with role $k-1$ and k, we then find that d has role $k-1$.

The above implies that a and b have their role in $\{k-2, k-1, k\}$. Suppose $k = 3$. If $r(a) = 1$, then $r(b) = 2$ and $r(u_1) = 1$ implying that r is a 2-role assignment (as $r(c) = r(d) = 2$). Suppose $r(a) = 2$. Then a needs a neighbor with role 1. Hence $r(b) = 1$, but then r is a 2-role assignment since $r(u_1) = 1$. Suppose $r(a) = 3$. Then $r(b) \neq 2$, as otherwise b needs a neighbor with role 1. Hence $r(b) = 3$ and $r(u_1) = 1$. This means that r is an R^*-role assignment. Suppose $k \geq 4$. If $r(a) = k-2$, then a needs a neighbor with role $k-3$. So, $r(b) = k-3$. However, this is not possible since vertex b with role $k-3$ is adjacent to vertex c with role $k-1$. If $r(a) = k-1$, then $r(b) = k-2$. This is not possible either, as then b would still need a neighbor with role $k-3$. Hence $r(a) = k$ and for the same reasons $r(b) = k$. Then r is an R^*-role assignment.

We claim that $(Q, \mathcal{S})$ has a 2-coloring if and only if G has a k-role assignment.

Suppose $(Q, \mathcal{S})$ has a 2-coloring (Q_1, Q_2). We show that G has an R^*-role assignment, which is a k-role assignment. We assign role i to each p_i^S

for $i = 1, \ldots, k-2$ and role $k-1$ to each $S \in \mathcal{S}$. As $(Q, \mathcal{S})$ is non-trivial, either Q_1 or Q_2, say Q_2, has size at least two. Then we assign role $k-1$ to each $q \in Q_1$ and role $k-2$ to neighbor u_1^q. We assign role k to each $q \in Q_2$ and $k-1$ to neighbor u_1^q. As $|Q_2| \geq 2$, every vertex in Q has a neighbor with role k. Hence, we can finish off the role assignment by assigning roles to the remaining vertices of each copy H^q of H as follows. For convenience, we remove the superscript q. We map each path S, T, T' to the path $1 \cdots k-2$, where $r(s_i) = r(t_i) = r(t'_i) = i$ for $i = 1, \ldots, k-2$. If u_1 received role $k-2$ we assign u_i role $k-1-i$ for $i = 2, \ldots, k-2$ and we assign u_{k-2+i} role $i+1$ for $i = 1, \ldots, k-2$. Furthermore, we assign role $k-1$ to c, d, w, and role k to a, b, x, y, z. If u_1 received role $k-1$, it already has a neighbor with role k (namely its neighbor in Q). Then we assign u_i role $k-i$ for $i = 2, \ldots, k-1$ and we assign u_{k-1+i} role $i+1$ for $i = 1, \ldots, k-3$. Furthermore, we assign role $k-1$ to c, d, w, x, and role k to a, b, y, z.

To prove the reverse statement, suppose G has a k-role assignment r. As we have shown above, by construction, G must have an R^*-role assignment. Then each p_i^S must have role i for $i = 1, \ldots, k-2$. Then $r(S) = k-1$ for each $S \in \mathcal{S}$, and each S must have a neighbor in Q with role $k-1$ and a neighbor in Q with role k. We define $Q_1 = \{q \in Q \mid r(q) = k-1\}$ and $Q_2 = Q \setminus Q_1$. Then we find that (Q_1, Q_2) is a 2-coloring of $(Q, \mathcal{S})$. This completes the proof of Theorem 5.8. □

5.4 Conclusion

We have settled an open problem of Sheng [249] by showing that it can be decided in $\mathcal{O}(n^2)$ time if a chordal graph has a k-role assignment when $k = 2$. We also showed that for any fixed $k \geq 3$ the k-Role Assignment problem remains NP-complete when restricted to chordal graphs.

Role assignments are also studied in topological graph theory. There, a graph G is called an *emulator* of a graph R if G has an R-role assignment. Then the question is which graphs allow planar emulators; see for example the recent paper by Rieck and Yamashita [228] for nice developments in this area. An interesting question is the computational complexity of the k-Role Assignment problem for planar graphs. The answer to this question is already unknown for $k = 2$.

Chapter 6

Finding induced paths of given parity in claw-free graphs

This chapter is based on the following paper.

[155] P. van 't Hof, M. Kamiński, and D. Paulusma. Finding induced paths of given parity in claw-free graphs. In: *Proceedings of the 35th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2009)*, volume 5911 of *Lecture Notes in Computer Science*, pages 341–352, Springer, 2009.

Finding a shortest path, a maximum independent set or a hamiltonian cycle in a graph are just a few examples from the wide spectrum of problems dealing with finding a subgraph (or an induced subgraph) with some particular property. In this context, very simple subgraphs, such as paths, trees and cycles, with some prescribed property are often studied. In this chapter and the next we consider two such problems: Chapter 7 deals with the problem of finding a longest cycle in a graph, while the problem of finding induced paths of odd or even length is the main focus of this chapter. Despite the fact that the structure of the subgraphs we are looking for is very simple, the problem of finding such subgraphs is NP-hard on general graphs. In both chapters we consider the case where the input graph is claw-free. Although the problem studied in Chapter 7 remains NP-hard under this restriction, we show that the problems studied in this chapter can be solved in polynomial time when restricted to the class of claw-free graphs. In order to obtain the polynomial

time algorithms presented in this chapter, we exploit several structural properties of claw-free graphs, including a structural characterization of claw-free perfect graphs due to Chvátal and Sbihi [76].

6.1 Background and results

The following problem has been extensively studied in the context of perfect graphs.

Parity Path
Instance: A graph G and two vertices s, t of G.
Question: Does G contain an odd and an even induced path from s to t?

We focus on the closely related problem of deciding whether there exists an induced path of given parity between a pair of vertices. In particular, we study the following two problems.

Odd Induced Path
Instance: A graph G and two vertices s, t of G.
Question: Does G contain an odd induced path from s to t?

Even Induced Path
Instance: A graph G and two vertices s, t of G.
Question: Does G contain an even induced path from s to t?

The Odd Induced Path problem was shown to be NP-complete by Bienstock [32]. Consequently, the Even Induced Path problem and the Parity Path problem are NP-complete as well. Several authors however have identified a number of graph classes that admit polynomial time algorithms for these problems. Below we survey those results, as well as results on related problems, before stating our contribution.

Odd Path and Even Path. In the Odd Path and Even Path problems the task is to find a (not necessarily induced) path of given parity between a specified pair of vertices. These problems were considered by LaPaugh and Papadimitriou [187]. They mention an $\mathcal{O}(n^3)$ time algorithm for solving both problems due to Edmonds, using a reduction to matching, and propose a faster one of $\mathcal{O}(m)$ time complexity. Their algorithm also finds a *shortest* (not necessarily induced) path of given parity between two vertices in $\mathcal{O}(m)$

time, even in a weighted graph. Interestingly, as they also show in their paper, the problem of finding a directed path of given parity is NP-complete for directed graphs.

Arkin, Papadimitriou and Yannakakis [12] generalized the result of [187] and designed a linear-time algorithm deciding if all (not necessarily induced) paths between two specified vertices are of length p mod q, for fixed integers p and q.

Even Pair. First interest in induced paths of given parity comes from the theory of perfect graphs. Two non-adjacent vertices are called an *even pair* if every induced path between them is even. The Even Pair problem is to decide if a given pair of vertices forms an even pair. The Even Pair problem is co-NP-complete due to Bienstock [32], as is the problem of deciding if a graph contains an even pair. The interest in even pairs was sparked by an observation of Fonlupt and Uhry [118]: if a graph is perfect and contains an even pair, then the graph obtained by identifying the vertices that form the even pair is also perfect. Later Meyniel [208] showed that minimal non-perfect graphs contain no even pair. Those two facts triggered a series of theoretical and algorithmic results which are surveyed in [100] and its updated version [101].

There is some evidence that perfect graphs without an even pair can be generated by performing a small number of composition operations on some basic graphs. Using such a structural result could then lead to a combinatorial algorithm for coloring perfect graphs. Indeed, for coloring perfect graphs using at most three colors this approach turned out to be successful, as was shown by Chudnovsky and Seymour in [74]. Linhares Sales and Maffray [197] study even pairs in order to give characterizations of claw-free graphs that are strict quasi-parity and perfectly contractile, respectively; a graph is *strict quasi-parity* if every induced subgraph is either a complete graph or contains an even pair, and a graph is *perfectly contractile* if every induced subgraph has a sequence of even pair contractions that leads to a complete graph.

Parity Path and Group Path. Arikati, in a series of papers with different coauthors, developed polynomial time algorithms for the Parity Path problem in different classes of graphs. Chordal graphs are considered in [9], where the authors present a linear-time algorithm for the Group

Path problem, a generalization of the Odd Induced Path problem. In the Group Path problem the edges of the input graph are weighted with elements of some group $\mathcal{G}$. The problem is to find an induced path of given weight between two specified vertices, where the weight of a path is defined as the product of the weights of the edges of the path. They present an $\mathcal{O}(|\mathcal{G}|m+n)$ time algorithm for the Group Path problem on chordal graphs using a perfect elimination ordering.

The topic of [11] is Parity Path on circular-arc graphs. The authors show how to reduce the problem to interval graphs by recursively applying a set of reductions. Since interval graphs are chordal, the algorithm of [9] can be used to obtain the solution. This way they obtain a polynomial time algorithm for circular-arc graphs. In [240] polynomial time algorithms for the Parity Path problem on comparability and cocomparability graphs, and a linear-time algorithm for permutation graphs are given. A polynomial time algorithm for Parity Path on perfectly orientable graphs is presented in [10]. Sampaio and Linhares Sales [239] obtain a polynomial time algorithm for planar perfect graphs. The authors of [110] characterize even and odd pairs in comparability and P_4-comparability graphs and give polynomial time algorithms for the Parity Path problem in those classes. Hoàng and Le [153] show that Parity Path can be solved in polynomial time for the class of 2-split graphs.

Note that a set F of vertices of a line graph $G = L(H)$ form an odd (respectively even) induced path in G if and only if the set of edges corresponding to F form an even (respectively odd) path in the preimage graph H of G. It is well-known that the preimage graph of a line graph can be found in polynomial time [237]. Combining these two facts with the polynomial time algorithm for finding (not necessarily induced) paths of given parity in [187] yields a polynomial time algorithm for solving the Parity Path problem for the class of line graphs (cf. [258]).

Our interest in the Odd Induced Path problem was in part stirred by studying Bienstock's NP-completeness reduction in [32]. He builds a graph out of a 3-Satisfiability formula and shows that the formula is satisfiable if and only if there exists an odd induced path between a certain pair of vertices. This is also shown to be equivalent to the existence of two disjoint induced paths (with no edges between the two paths) between certain pairs of vertices

in the construction. Finding such two paths is then NP-hard in general but has been proved solvable in polynomial time for claw-free graphs [192]. A natural question to ask is whether the Odd Induced Path problem can also be solved in polynomial time for this class of graphs. In Section 6.3 we answer this question in the affirmative by presenting an algorithm that solves both the Odd Induced Path problem and the Even Induced Path problem in $\mathcal{O}(n^5)$ time for the class of claw-free graphs. This implies that the Even Pair problem can be solved in $\mathcal{O}(n^5)$ time for claw-free graphs.

As we saw earlier in this section, the Parity Path problem has been extensively studied in different graph classes. However, a polynomial time algorithm for claw-free graphs has never been proposed; somewhat surprising, since claw-free graphs form a large and important class containing, e.g., the class of line graphs, the class of complements of triangle-free graphs, and the class of proper interval graphs (see Section 1.2.4 for more information on claw-free graphs). Our $\mathcal{O}(n^5)$ time algorithm for solving the Odd Induced Path and Even Induced Path problems for claw-free graphs immediately implies that we can solve the Parity Path problem for claw-free graphs in $\mathcal{O}(n^5)$ time, thus generalizing the aforementioned polynomial time result on line graphs. Making use of the structure of claw-free perfect graphs we also obtain an $\mathcal{O}(n^7)$ time algorithm for finding *shortest* induced paths of given parity between two specified vertices in a claw-free perfect graph. This algorithm is presented in Section 6.4.

Apart from the Odd Induced Path problem, Bienstock [32] mentioned two more NP-complete problems in the abstract of his paper. The first one is to decide whether a graph has an odd hole passing through a given vertex. The second one is to decide whether a graph has an odd induced path between *every* pair of vertices. We show in Section 6.3 that our polynomial time algorithm for the Odd Induced Path problem implies that both these problems are solvable in $\mathcal{O}(n^7)$ time when restricted to the class of claw-free graphs. The same holds for the problem of deciding whether or not a claw-free graph contains an even pair.

Several times in this chapter we make use of a fascinating structural characterization of claw-free perfect graphs due to Chvátal and Sbihi [76], who used this characterization to obtain a polynomial time recognition algorithm for this graph class. We present the key ideas behind this recognition algo-

rithm in Section 6.2, where we also perform a precise running time analysis. The reason for this is the fact that Chvátal and Sbihi's algorithm is used as a subroutine in the polynomial time algorithm presented in Section 6.3, and its key ideas will be used to obtain the results in Section 6.4.

6.2 Recognizing claw-free perfect graphs in $\mathcal{O}(n^4)$ time

A graph is called *Berge* if it does not contain an odd hole or an odd antihole. A little over 40 years after Berge [25] conjectured that a graph is perfect if and only if it is Berge, Chudnovsky et al. [64] confirmed his intuition by proving the following theorem.

Theorem 6.1 (Strong Perfect Graph Theorem, [64])**.** *A graph is perfect if and only if it contains no odd hole and no odd antihole.*

Shortly afterwards, Chudnovsky et al. [62] presented an $\mathcal{O}(n^9)$ time algorithm for recognizing Berge graphs. This means we can determine in $\mathcal{O}(n^9)$ time whether or not a graph is perfect. The main goal of this section is to show that the problem of deciding whether or not a *claw-free* graph is perfect can be solved in $\mathcal{O}(n^4)$ time. We point out that we do not actually present a new algorithm for recognizing claw-free perfect graphs, but merely perform an exact running time analysis of an existing recognition algorithm due to Chvátal and Sbihi [76]. Chvátal and Sbihi did not explicitly state the time complexity of their recognition algorithm, and to the best of our knowledge no better upper bound on the time needed to recognize claw-free perfect graphs than the aforementioned $\mathcal{O}(n^9)$ can be found in the literature. We will use the recognition algorithm for claw-free perfect graphs as a subroutine in the algorithm presented in Section 6.3. Moreover, in Section 6.4 we will exploit the fascinating structure of perfect claw-free graphs exhibited by Chvátal and Sbihi [76] in order to obtain an algorithm for finding shortest induced paths of given parity in such graphs. Before we present the key ideas behind Chvátal and Sbihi's algorithm, we need to introduce some terminology and techniques they use in their paper.

A set $X \subseteq V(G)$ is a *clique separator* of a connected graph G if X is a separator of G that is a clique. Suppose that the graph obtained from G by

deleting X consists of k connected components with vertex sets $V_1, \ldots, V_k$. We call the graphs $G[V_1 \cup X], \ldots, G[V_k \cup X]$ the *children* of G produced by X. If any child $G[V_i \cup X]$ contains a clique separator X_i, we continue decomposing the graph G by replacing $G[V_i \cup X]$ by the children of $G[V_i \cup X]$ produced by X_i. Repeating this procedure until the graph cannot be decomposed any further yields a collection $\mathcal{C} = \{G_1, \ldots, G_p\}$ of induced subgraphs of G without a clique separator, called the *atoms* of G. We refer to the set $\mathcal{C}$ as a *clique separator decomposition* of G. Several authors designed polynomial time algorithms for finding a clique separator decomposition of a graph, the fastest one being due to Tarjan [254]. We give an explicit description of Tarjan's algorithm in Section 6.4, where we also prove some properties about the obtained clique separator decomposition. These properties are then used in the proof of the main result of that section. For now, we only mention the following result.

Theorem 6.2 ([254])**.** *Given a connected graph G, a clique separator decomposition of G containing at most $n - 1$ atoms can be found in $\mathcal{O}(nm)$ time.*

According to Whitesides [262], the original motivation to study clique separator decompositions is their relation to the problem of recognizing perfect graphs.

Theorem 6.3 ([262])**.** *A graph is perfect if and only if all its atoms are perfect.*

Chvátal and Sbihi [76] discovered that all atoms in a clique separator decomposition of a claw-free perfect graph G belong to one of two classes of graphs, which they called "elementary" and "peculiar". We now give the definitions of elementary and peculiar graphs, and show that we can determine in $\mathcal{O}(n^3)$ time whether a graph belongs to one of those classes.

Definition 6.4. *A graph H is* elementary *if its edges can be colored with two colors such that every induced path on three vertices has its two edges colored differently. We call such a coloring an* elementary coloring *of H.*

See Figure 6.1 for an example of an elementary graph with an elementary coloring, where the light edges and heavy edges are colored differently. This

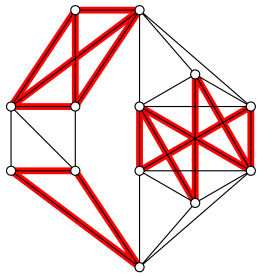

Figure 6.1: An elementary graph with an elementary coloring.

graph was presented as an example of an elementary graph in [205]. The authors of [76] describe how elementary graphs can be recognized in polynomial time. Using their arguments, it is easy to prove the time complexity in the following lemma.

Lemma 6.5 ([76])**.** *It is possible to decide in $\mathcal{O}(n^3)$ time whether or not a graph H is elementary. If it is, an elementary coloring of H can be found in $\mathcal{O}(n^3)$ time.*

Proof. Let H be a graph on n vertices and m edges. We construct a graph $\Gamma(H)$ on m vertices as follows: $V(\Gamma(H)) = E(H)$ and two vertices in $\Gamma(H)$ are adjacent if and only if the corresponding edges in G induce a path on three vertices. It is clear that we can compute the graph $\Gamma(H)$ in $\mathcal{O}(n^3)$ time by checking for each triple of vertices in $V(H)$ whether they induce a path on three vertices in H. For any edge uv of H, there are at most n vertices of H that are adjacent to exactly one vertex of $\{u, v\}$, which means that the graph $\Gamma(H)$ has at most nm edges. It is easy to see that H is elementary if and only if $\Gamma(H)$ is bipartite, and any 2-coloring of $\Gamma(H)$ corresponds to an elementary coloring of H. Clearly, finding a 2-coloring of a graph with m vertices and nm edges, or concluding that such a coloring does not exist, can be done in $\mathcal{O}(m + nm) = O(n^3)$ time. □

Definition 6.6. *A graph H is* peculiar *if it can be obtained from a complete graph K as follows. Partition $V(K)$ into six mutually disjoint non-empty sets A_i, B_i, $i = 1, 2, 3$. For each $i = 1, 2, 3$, remove at least one edge with one end-vertex in A_i and the other end-vertex in B_{i+1}, where the subscript 4 is interpreted as 1. Finally, add three new mutually disjoint non-empty*

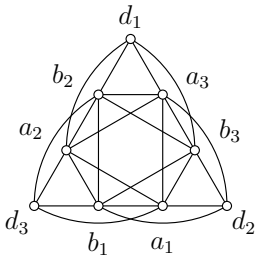

Figure 6.2: The smallest possible peculiar graph.

complete graphs D_i, $i = 1, 2, 3$, and for each $i = 1, 2, 3$ make each vertex in D_i adjacent to all vertices in $V(K) \setminus (A_i \cup B_i)$.

The smallest possible peculiar graph is depicted in Figure 6.2. Li and Zang [196] present a simple polynomial time algorithm for recognizing peculiar graphs.

Lemma 6.7 ([196])**.** *It is possible to decide in $\mathcal{O}(n^3)$ time whether or not a graph is peculiar.*

It is not hard to verify that both elementary graphs and peculiar graphs cannot contain an odd hole or an odd antihole (cf. [205]), which means they are perfect by virtue of the Strong Perfect Graph Theorem [64]. As mentioned before, Chvátal and Sbihi [76] proved that every atom of a clique separator decomposition of any claw-free perfect graph is either elementary or peculiar. This means we can formulate their main result as follows.

Theorem 6.8 ([76])**.** *A claw-free graph G with no clique separator is perfect if and only if it is either elementary or peculiar.*

Using the explicit time complexities of the recognition algorithms for elementary and peculiar graphs in Lemma 6.5 and Lemma 6.7, we can determine the time complexity of the recognition algorithm for claw-free perfect graphs by Chvátal and Sbihi [76].

Theorem 6.9 ([76])**.** *It is possible to decide in $\mathcal{O}(n^4)$ time whether or not a claw-free graph is perfect.*

Proof. Let G be a claw-free perfect graph. To test whether or not G is perfect, we act as follows. First we find a clique separator decomposition

$\mathcal{C} = \{G_1, \ldots, G_p\}$ of G, which we can do in $\mathcal{O}(n^3)$ time by Theorem 6.2. Since every atom $G_i \in \mathcal{C}$ is a claw-free graph without a clique separator, G_i is perfect if and only if G_i is elementary or peculiar by Theorem 6.8. Lemma 6.5 and Lemma 6.7 together imply that for each atom G_i we can decide in $\mathcal{O}(n^3)$ time whether G_i is elementary, peculiar, or neither. By Theorem 6.3, G is perfect if and only if every atom $G_i \in \mathcal{C}$ is perfect. Since we only have to consider at most $n-1$ atoms by Theorem 6.2, this yields an overall time complexity of $\mathcal{O}(n^4)$. □

Corollary 6.10. *Let G be a claw-free graph. It is possible to find an odd hole or an odd antihole of G, or conclude that such a subgraph does not exist, in $\mathcal{O}(n^5)$ time.*

Proof. Let G be a claw-free graph. We test whether or not G is perfect, which we can do in $\mathcal{O}(n^4)$ time by Theorem 6.9. By the Strong Perfect Graph Theorem, G only contains an odd hole or an odd antihole if G is not perfect. In that case, we remove a vertex from G and check in $\mathcal{O}(n^4)$ time if the obtained graph G' is perfect. If so, we restore the vertex and repeat the procedure on G, removing another vertex. If not, we repeat the whole procedure on the smaller graph G'. By repeating this procedure as long as possible, we find a minimal imperfect induced subgraph H of G. (A graph is called *minimal imperfect* if it is not perfect, but all its proper induced subgraphs are perfect.) By the Strong Perfect Graph Theorem, H is an odd hole or an odd antihole of G. The $\mathcal{O}(n^5)$ overall time complexity follows from the fact that we have to apply the $\mathcal{O}(n^4)$ time recognition algorithm for claw-free perfect graphs $\mathcal{O}(n)$ times. □

6.3 Finding induced paths of given parity

In this section we present an algorithm that solves the Odd Induced Path problem in $\mathcal{O}(n^5)$ time for claw-free graphs. We show that, apart from solving the decision problem, it is also possible to *find* an odd induced path between two given vertices of a claw-free graph, or conclude that such a path does not exist, in $\mathcal{O}(n^5)$ time. Here is an outline of our algorithm.

Algorithm solving Odd Induced Path **for claw-free graphs**

Input	:	claw-free graph G, vertices s and t of G
Output	:	`YES` if G contains an odd induced path from s to t
		`NO` otherwise

Preprocess G to obtain graph G''

Step 1: add edges to make s and t simplicial

Step 2: delete irrelevant vertices

Test whether or not G'' is perfect

If G'' is not perfect, output `YES`

If G'' is perfect, find a shortest path P from s to t

If P is odd, output `YES`

If P is even, define graph $G^* := (V(G'') \cup \{x\}, E(G'') \cup \{sx, tx\})$

Test whether or not G^* is perfect

If G^* is not perfect, output `YES`

If G^* is perfect, output `NO`

As shown in the outline, we first preprocess the input graph G in order to obtain a new graph G'' with certain desirable properties. This preprocessing procedure is described in Section 6.3.1. We then distinguish two cases, depending on whether or not G'' is perfect. The case that G'' is not perfect is discussed in Section 6.3.2, while Section 6.3.3 deals with the case that G'' is perfect. In Section 6.3.4 we prove correctness of our algorithm and show that its time complexity is $\mathcal{O}(n^5)$. We also explain in Section 6.3.4 how our algorithm can be slightly modified in such a way that it also solves the Even Induced Path problem for claw-free graphs in $\mathcal{O}(n^5)$ time.

6.3.1 Preprocessing the input graph G

Let G be a claw-free graph and let s and t be two vertices of G. Note that we may without loss of generality assume that G is connected and that s and t are not adjacent. We make these assumptions throughout the chapter.

Step 1. We add an edge between each pair of non-adjacent neighbors of s, and we do the same for each pair of non-adjacent neighbors of t. Then

in the resulting graph G', both s and t are *simplicial* vertices, i.e., vertices whose neighborhood form a clique in G'. In general, adding edges is not a claw-freeness preserving operation. However, the following lemma states that we do not create claws in Step 1.

Lemma 6.11. *The graph G' is claw-free.*

Proof. Suppose, for contradiction, that G' contains an induced subgraph isomorphic to a claw. Let $K := \{x, a, b, c\}$ be a set of vertices of G' inducing a claw with center x. Note that the fact that s is simplicial implies $x \neq s$. Since G is claw-free, we may without loss of generality assume that at least two vertices of K must be in $N_{G'}(s) \cup \{s\}$. Since $N_{G'}(s) \cup \{s\}$ is a clique of G' and $\{a, b, c\}$ is an independent set of G', we may without loss of generality assume that $K \cap (N_{G'}(s) \cup \{s\}) = \{x, a\}$ and $\{b, c\} \subseteq V(G') \setminus (N_{G'}(s) \cup \{s\})$. Then $\{x, b, c, s\}$ induces a claw in G with center x, a contradiction. □

Step 2. We "clean" G' by repeatedly deleting irrelevant vertices. A vertex $v \in V(G')$ is called *irrelevant* (for vertices s and t) if v does not lie on any induced path from s to t, and we say that G' is *clean* (for s and t) if none of its vertices is irrelevant. Let G'' denote the graph obtained from G' by repeatedly deleting vertices that are irrelevant. Note that G'' is claw-free, as G'' is an induced subgraph of G'.

We now show that we can perform Step 2 in polynomial time by showing that we can identify irrelevant vertices in polynomial time. In general, the problem of deciding whether a vertex is irrelevant is NP-complete. This follows from a result by Derhy and Picouleau [90], who prove that the following problem is NP-complete for the class of graphs of maximum degree at most 3.

Three-in-a-Path
Instance: A graph G and three vertices v_1, v_2, v_3 of G.
Question: Does there exist an induced path of G containing v_1, v_2 and v_3?

Chudnovsky and Seymour [73] study the following closely related problem.

Three-in-a-Tree
Instance: A graph G and three vertices v_1, v_2, v_3 of G.
Question: Does there exist an induced tree of G containing v_1, v_2 and v_3?

Theorem 6.12 ([73]). *The* Three-in-a-Tree *problem can be solved in* $\mathcal{O}(n^4)$ *time, and a desired tree can be found in* $\mathcal{O}(n^4)$ *time in case one exists.*

Observe that the Three-in-a-Path problem is equivalent to the Three-in-a-Tree problem for the class of claw-free graphs, since every induced tree in a claw-free graph is an induced path. Hence, using Theorem 6.12, we can prove the following result.

Lemma 6.13. *The problem of deciding whether a vertex* v *of a claw-free graph* G *is irrelevant for two simplicial vertices* s *and* t *of* G *can be solved in* $\mathcal{O}(n^4)$ *time.*

Proof. We claim that there exists an induced path in G from s to t containing v if and only if G together with s, t, v is a yes-instance of the Three-in-a-Tree problem. By Theorem 6.12, this proves that we can decide in $\mathcal{O}(n^4)$ time if v is irrelevant.

If there exists a path in G from s to t containing v, then that path is an induced tree containing all three vertices. Now suppose that there exists an induced subgraph T of G which is a tree containing s, t and v. Recall that any induced subgraph of a claw-free graph which is a tree is in fact an induced path. Since vertices s and t are simplicial, any induced path containing s and t contains exactly one neighbor of s and one neighbor of t. Hence s and t must be the endpoints of the path T. □

After preprocessing the input graph G we have obtained a graph G'' that satisfies the following three conditions:

(1) G'' is claw-free;
(2) both s and t are simplicial vertices of G'';
(3) G'' is clean for s and t.

The following lemma implies that solving the Odd Induced Path problem for G is equivalent to solving the problem for G''. The lemma also shows that the entire preprocessing procedure can be performed in $\mathcal{O}(n^5)$ time.

Lemma 6.14. *Every induced path from* s *to* t *in* G'' *is also an induced path from* s *to* t *in* G*, and vice versa. Moreover,* G'' *can be obtained from* G *in* $\mathcal{O}(n^5)$ *time.*

Proof. It is clear that by adding edges in Step 1 no new induced path from s to t is created. Since any induced path from s to t in G contains exactly one vertex of $N_G(s)$ and exactly one vertex of $N_G(t)$, the graph G' obtained after Step 1 contains all induced paths from s to t that were contained in G. In Step 2, we only remove vertices that do not lie on any induced path from s to t. This implies that every induced path from s to t in G'' is also an induced path from s to t in G, and vice versa. It is clear that we can perform Step 1 in $\mathcal{O}(n^2)$ time. In Step 2, we have to check for $\mathcal{O}(n)$ vertices whether or not they are irrelevant. Since we can do this in $\mathcal{O}(n^4)$ time per vertex by Lemma 6.13, we can perform Step 2, and consequently the entire preprocessing procedure, in $\mathcal{O}(n^5)$ time. □

We now distinguish two cases, depending on whether or not G'' is perfect.

6.3.2 G'' is not perfect

Suppose G'' is not perfect. Then G'' contains an odd hole or an odd antihole by virtue of the Strong Perfect Graph Theorem. We consider odd antiholes and odd holes in Lemma 6.15 and Lemma 6.16, respectively. Recall that the length of an antihole is the number of edges in its complement.

Lemma 6.15. *Let H be a connected claw-free graph. If H contains a simplicial vertex, then H does not contain an odd antihole of length more than 5.*

Proof. Let s be a simplicial vertex of a connected claw-free graph H. For contradiction, suppose H contains an odd antihole X such that its complement $\overline{X} = x_1x_2 \dots x_{2k+1}x_1$ is an odd induced cycle with $k \geq 3$. Vertex s does not belong to X, since s is simplicial. Let P be an induced path from s to a vertex of X such that $|V(P)|$ is minimum. Note that such a path P exists since H is connected. Without loss of generality assume that $V(P) \cap V(X) = \{x_1\}$.

Let s' be the neighbor of x_1 on P. We claim that s' is adjacent to at most one vertex of $\{x_i, x_{i+1}\}$ for $1 \leq i \leq 2k$. If $s' = s$, this claim immediately follows from the assumption that s is simplicial and the fact that x_i and x_{i+1} are not adjacent. Suppose $s' \neq s$, and let s'' be the neighbor of s' on P not equal to x_1. Note that s'' is not adjacent to any vertex of X due to

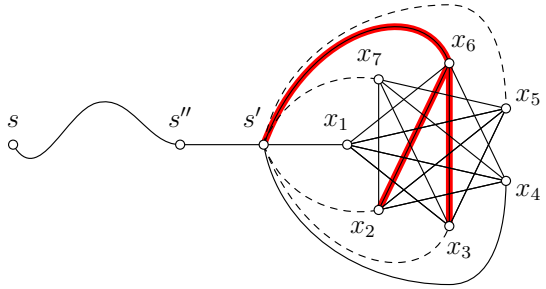

Figure 6.3: A claw induced by $\{x_6, s', x_2, x_3\}$ with center x_6.

the minimality of $|V(P)|$. Vertex s' cannot be adjacent to both x_i and x_{i+1}, since then the set $\{s', s'', x_i, x_{i+1}\}$ induces a claw in H with center s'. Hence s' is adjacent to at most one vertex of $\{x_i, x_{i+1}\}$ for $1 \leq i \leq 2k$.

Note that vertex s' is adjacent to at least one vertex of $\{x_i, x_{i+1}\}$ for $3 \leq i \leq 2k-1$, as otherwise $\{x_1, s', x_i, x_{i+1}\}$ induces a claw in H with center x_1. This, together with the fact that s' is adjacent to at most one vertex of $\{x_i, x_{i+1}\}$ for $1 \leq i \leq 2k$, implies that s' is adjacent to exactly one vertex of $\{x_3, x_{2k}\}$. Without loss of generality, assume that s' is adjacent to x_{2k} and not to x_3. Since s' is adjacent to x_1 and s' is adjacent to at most one vertex of $\{x_i, x_{i+1}\}$ for $1 \leq i \leq 2k$, s' is not adjacent to x_2. Note that x_3 is adjacent to x_{2k}, since $k \geq 3$. But then $\{x_{2k}, s', x_2, x_3\}$ induces a claw in H with center x_{2k}; see Figure 6.3 for an illustration of the case where $k = 3$. This contradiction finishes the proof of Lemma 6.15. □

We point out that the arguments in the proof of Lemma 6.15 can also be used to prove that every odd antihole X of a connected claw-free graph H is dominating, i.e., every vertex of H either belongs to X or has a neighbor in X.

Lemma 6.16. *Let H be a connected claw-free graph that is clean for two simplicial vertices s and t. If H contains an odd hole, then there exists both an odd and an even induced path from s to t.*

Proof. Let C be an odd hole of H. Let P be an induced path from s to a vertex p of C and let Q be an induced path from t to a vertex q of C, such that there is no edge in H connecting a vertex in $P[V(P) \setminus \{p\}]$ to a vertex

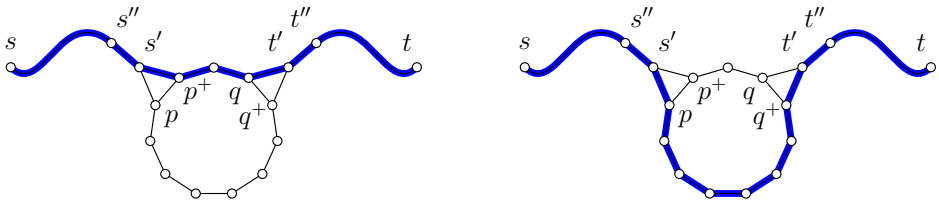

Figure 6.4: Two induced paths from s to t of different parity.

in $Q[V(Q) \setminus \{q\}]$ and such that $|V(P)| + |V(Q)|$ is minimum. Note that such paths P and Q exist since H is clean and connected. Let s' be the neighbor of p on P, and let t' be the neighbor of q on Q; we note that possibly $s' = s$ and $t' = t$.

Claim 1. *Both s' and t' are adjacent to exactly two adjacent vertices of C.*

Suppose p is the only vertex of C that is adjacent to s'. Let p^- (respectively p^+) denote the neighbor of p on C when we traverse C in counter-clockwise (respectively clockwise) order. The set $\{p, p^-, p^+, s'\}$ induces a claw in H with center p, contradicting the claw-freeness of H. Hence s' must be adjacent to at least one vertex of $\{p^-, p^+\}$. Suppose there exists a set $D \subseteq V(C)$ such that $|D| \geq 3$ and s' is adjacent to every vertex in D. Since C is an induced cycle, we know that D contains two vertices d_1 and d_2 that are not adjacent. Since s is simplicial and therefore does not have two non-adjacent neighbors, we must have $s' \neq s$. Let $s'' \neq p$ be a neighbor of s' on P; possibly $s'' = s$. Vertex s'' is not adjacent to any vertex of C due to the minimality of $|V(P)| + |V(Q)|$, which means the set $\{s', d_1, d_2, s''\}$ induces a claw in H with center s'. This contradiction finishes the proof of Claim 1 for vertex s'. By symmetry the claim also holds for vertex t'.

We assume, without loss of generality, that $N_H(s') \cap V(C) = \{p, p^+\}$ and $N_H(t') \cap V(C) = \{q, q^+\}$. We distinguish three cases.

Suppose $|\{p, p^+\} \cap \{q, q^+\}| = 0$. Since C is an odd hole, the induced path $s'p^+\overrightarrow{C}qt'$ and the induced path $s'p\overleftarrow{C}q^+t'$ have different parity. Since by definition there is no edge connecting a vertex in $P[V(P) \setminus \{p\}]$ to a vertex in $Q[V(Q) \setminus \{q\}]$, this means there exists both an odd and an even induced path from s to t in H; see Figure 6.4 for an illustration.

Suppose $|\{p, p^+\} \cap \{q, q^+\}| = 1$. Without loss of generality, suppose

$p^+ = q$. Then the path $s'qt'$ is an even induced path from s' to t', and the path $s'p\overleftarrow{C}q^+t'$ is an odd induced path from s' to t'. Since by definition there is no edge connecting a vertex in $P[V(P) \setminus \{p\}]$ to a vertex in $Q[V(Q) \setminus \{q\}]$, this means there exists both an odd and an even induced path from s to t in H.

Suppose $|\{p, p^+\} \cap \{q, q^+\}| = 2$. By Claim 1, neither s' nor t' is adjacent to p^-. Since s' and t' are not adjacent by the choice of P and Q, the set $\{p, p^-, s', t'\}$ induces a claw in H with center p. This contradiction finishes the proof. □

Recall that G'' is not perfect and has two simplicial vertices s and t. This, together with Lemma 6.15 and Lemma 6.16, implies that G'' contains both an odd and an even induced path from s to t. We now show that we can also *find* such paths in $\mathcal{O}(n^5)$ time.

Lemma 6.17. *If G'' is not perfect, then it is possible to find both an odd and an even induced path from s to t in G'' in $\mathcal{O}(n^5)$ time.*

Proof. Since G'' has two simplicial vertices s and t, G'' does not contain an odd antihole of length more than 5 by Lemma 6.15. Since an odd antihole of length 5 is also an odd hole of length 5, G'' contains an odd hole by virtue of the Strong Perfect Graph Theorem. We can find such a hole C in $\mathcal{O}(n^5)$ time by Corollary 6.10. Let c be any vertex of C, and let P be an induced path in G'' from s to t containing c. Note that such a path P exists since G'' is clean for s and t. We can find P in $\mathcal{O}(n^4)$ time as a result Theorem 6.12. It is clear from the proof of Lemma 6.16 that we can use P to find both an odd and an even induced path from s to t in G''. □

6.3.3 G'' is perfect

Suppose G'' is perfect. In the concluding remarks of their paper, Corneil and Fonlupt [81] pointed out that a polynomial time recognition algorithm for perfect graphs implies a polynomial time algorithm for the Parity Path problem for the class of perfect graphs. Interestingly, the arguments they used to prove this implication were already mentioned by Hsu [166] in the paper in which he introduced the Parity Path problem. Using their arguments, we can prove the following lemma.

Lemma 6.18. *If G'' is perfect, then it is possible to find an odd induced path from s to t in G'', or conclude that such a path does not exist, in $\mathcal{O}(n^5)$ time.*

Proof. Let P be a shortest path from s to t in G''. If P has odd length, then we are done. Suppose P has even length. Let G^* be the graph obtained from G by adding a vertex x and edges sx and tx. Note that the graph G^* is claw-free, since s and t are simplicial vertices of G''. We determine whether or not G^* is perfect, which we can do in $\mathcal{O}(n^4)$ time by Theorem 6.9. If G^* is perfect, then G^* does not contain an odd hole or an odd antihole by virtue of the Strong Perfect Graph Theorem. This means that all induced paths from s to t must be even, so we conclude that there does not exist an odd induced path from s to t. Suppose G^* is not perfect. Then G^* must contain an odd hole or an odd antihole, and vertex x must be in this odd hole or odd antihole since G is perfect. Since x has degree two, G^* cannot contain an odd antihole. Hence G^* contains an odd hole. We can find an odd hole C of G^* in $\mathcal{O}(n^5)$ time by Corollary 6.10. The graph obtained from C by removing vertex x is an odd induced path from s to t in G''. □

6.3.4 Finding induced paths of given parity from s to t in G

We are now ready to prove the main result of this section.

Theorem 6.19. *Both the* Odd Induced Path *problem and the* Even Induced Path *problem can be solved in $\mathcal{O}(n^5)$ time for the class of claw-free graphs. Moreover, an induced path from s to t of given parity can be found in $\mathcal{O}(n^5)$, if one exists.*

Proof. Let G be a claw-free graph, and let s and t be two vertices of G. Recall that we may without loss of generality assume that G is connected and that s and t are not adjacent. We preprocess G in $\mathcal{O}(n^5)$ time as described in Section 6.3.1, thus obtaining a graph G''. Recall that G'' is claw-free, that s and t are simplicial vertices in G'', and that G'' is clean for s and t. We test whether or not G'' is perfect, which we can do in $\mathcal{O}(n^4)$ time by Theorem 6.9. Below we show that we can find an induced path of given parity from s to t in G'', or conclude that such a path does not exist, in $\mathcal{O}(n^5)$ time. Lemma 6.14 implies that this suffices to prove Theorem 6.19.

If G'' is not perfect, then we can find both an odd and an even induced path from s to t in G'' in $\mathcal{O}(n^5)$ time by Lemma 6.17. If G'' is perfect, then

we can find an odd induced path from s to t in G'', or conclude that such a path does not exist, in $\mathcal{O}(n^5)$ time by Lemma 6.18. In order to find an even induced path from s to t, we define the graph G^* as the graph obtained from G'' by adding the edge st. It is easy to verify that adding the edge st creates neither a claw nor an odd antihole. Hence the arguments used in the proof of Lemma 6.18 can also be used to find an even induced path from s to t in G'', or conclude that such a path does not exist, in $\mathcal{O}(n^5)$ time. □

Theorem 6.19 immediately implies the following.

Corollary 6.20. *Both the* Parity Path *problem and the* Even Pair *problem can be solved in $\mathcal{O}(n^5)$ time for the class of claw-free graphs.*

Bienstock [32] proved that it is NP-complete to decide if a graph contains an odd induced path between *every* pair of vertices, which is equivalent to deciding if a graph contains no even pair. The following corollary of Theorem 6.19 implies that this problem can be solved in polynomial time when restricted to the class of claw-free graphs.

Corollary 6.21. *Deciding whether or not a claw-free graph has an even pair can be done in $\mathcal{O}(n^7)$ time.*

Proof. Let G be a claw-free graph. For each pair s, t of vertices of G, we can check in $\mathcal{O}(n^5)$ time whether or not they form an even pair by Corollary 6.20. Hence we can decide whether or not G has an even pair by performing this check $\mathcal{O}(n^2)$ times, each time with a different pair of vertices of G in the input. □

Another problem Bienstock [32] proved to be NP-complete is the problem of deciding whether a graph contains an odd hole passing through a prescribed vertex. The following corollary, which clearly also holds in case we are looking for an *even* hole, shows that this problem becomes polynomial time solvable when restricted to claw-free graphs.

Corollary 6.22. *It is possible to find an odd hole passing through a prescribed vertex of a claw-free graph, or conclude that such a hole does not exist, in $\mathcal{O}(n^7)$ time.*

Proof. Let G be a claw-free graph and let v be a vertex of G. We can find an odd hole of G passing through v, or conclude that such a hole does not exist, as follows. For each pair s, t of non-adjacent neighbors of v, let $G_{s,t}$ denote the (claw-free) graph obtained from G by removing v and all its neighbors, apart from s and t, from G. Clearly, G contains an odd hole through v if and only if the graph $G_{s,t}$ contains an odd induced path from s to t for some pair of non-adjacent neighbors s, t of v. We can find such a path, or conclude that such a path does not exist, in $\mathcal{O}(n^5)$ time by Theorem 6.19. The time complexity of $\mathcal{O}(n^7)$ follows from the fact that we have to perform our $\mathcal{O}(n^5)$ time algorithm for $\mathcal{O}(n^2)$ pairs of non-adjacent neighbors of v. □

6.4 Finding shortest induced paths of given parity

In this section we show that it is possible to find a *shortest* induced path of given parity between two specified vertices of a claw-free perfect graph in polynomial time, in case such a path exists. More specifically, we show that we can solve the following two problems in $\mathcal{O}(n^7)$ time for the class of claw-free perfect graphs.

Shortest Odd Induced Path
Instance: A graph G and two vertices s, t of G.
Task: Find a shortest odd induced path from s to t in G, or conclude that such a path does not exist.

Shortest Even Induced Path
Instance: A graph G and two vertices s, t of G.
Task: Find a shortest even induced path from s to t in G, or conclude that such a path does not exist.

Note that a shortest odd induced path between vertices s and t of a graph G is not necessarily a shortest odd path between s and t in G. For example, the shortest odd induced path from s to t in the graph in Figure 6.5 has length 5, whereas the shortest odd path from s to t has length 3.

Unlike the results in the previous section, we do not rely on the recognition algorithm for claw-free graphs that was described in Section 6.2 to prove the main result of this section. Instead, we make use of the structural properties of claw-free perfect graphs that were presented by Chvátal and

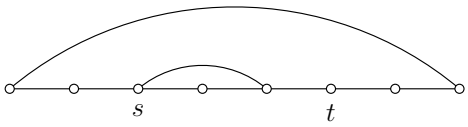

Figure 6.5: Shortest odd path from s to t is not shortest odd induced path.

Sbihi in [76]. Recall that they showed that a claw-free perfect graph with no clique separator is either elementary or peculiar (see Theorem 6.8). Below we first show in Section 6.4.1 that both the Shortest Odd Induced Path problem and the Shortest Even Induced Path problem can be solved in $\mathcal{O}(n^4)$ time for elementary graphs, and in $\mathcal{O}(n^3)$ time for peculiar graphs. In Section 6.4.2 we then present in detail Tarjan's clique separator decomposition algorithm that was already mentioned in Section 6.2. Finally, we prove in Section 6.4.3 that the Shortest Odd Induced Path and Shortest Even Induced Path problems can be solved in $\mathcal{O}(n^7)$ time for the class of claw-free perfect graphs.

6.4.1 Shortest paths in elementary and peculiar graphs

Let us start by showing how to find shortest induced paths of given parity in elementary graphs. Recall that a graph is elementary if and only if its edges can be colored with two colors such that every induced P_3 has both its edges colored differently.

Lemma 6.23. *Both the* Shortest Odd Induced Path *problem and the* Shortest Even Induced Path *problem can be solved in* $\mathcal{O}(n^4)$ *time for the class of elementary graphs.*

Proof. Let H be an elementary graph, and let u and v be two vertices of H. Note that we may assume, without loss of generality, that H is connected and that u and v are not adjacent. Suppose u and v have a common neighbor w. The even induced path uwv is the only induced path from u to v that contains w; in particular, w cannot lie on an odd induced path from u to v. Hence we may assume that u and v do not have a common neighbor.

We observe that any induced path from u to v in H contains exactly one vertex from $N_H(u)$ and exactly one vertex from $N_H(v)$. We also observe that in any elementary coloring of H, any two consecutive edges of any induced

path will be colored differently. Hence if there exists an odd (respectively even) induced path from u to v, then the first and the last edge of that path have the same color (respectively different colors). Using these observations, we can find a shortest odd induced path from u to v in H as follows.

We first find an elementary coloring $\varphi : E(H) \to \{0, 1\}$ of H; we can find such a coloring φ in $\mathcal{O}(n^3)$ time by Lemma 6.5. For every pair $u' \in N_H(u)$ and $v' \in N_H(v)$ with $\varphi(uu') = \varphi(vv')$, we define $H_{u'v'}$ to be the graph obtained from H by deleting the set $(N_H(u) \cup N_H(v) \cup \{u, v\}) \setminus \{u', v'\}$. Note that $H_{u'v'}$ is well-defined, since u and v are not adjacent and have no common neighbors. We either find a shortest path P' from u' to v' in $H_{u'v'}$, or conclude that such a path does not exist. It is well-known that we can do this in $\mathcal{O}(n^2)$ time. If there exists a shortest path P' from u' to v', then this path P' is clearly an induced path in $H_{u'v'}$. We add the vertices u and v as well as the edges uu' and vv' to P', which yields an induced path P from u to v in H. Since P is induced and φ is an elementary coloring, the colors 0 and 1 alternate on P. Then P is an odd induced path from u to v in H, since $\varphi(uu') = \varphi(vv')$. By performing this procedure for all pairs u', v' with $\varphi(uu') = \varphi(vv')$, we either find a shortest odd induced path from u to v in H, or conclude that such a path does not exist. It is clear that the procedure can be executed in $\mathcal{O}(n^4)$ time.

To solve the Shortest Even Induced Path problem, we perform the above procedure for all pairs u', v' with $\varphi(uu') \neq \varphi(vv')$ instead of $\varphi(uu') = \varphi(vv')$. □

It is clear from Definition 6.6 that the vertex set of every peculiar graph can be partitioned into nine disjoint cliques. Since every induced path contains at most two vertices of any clique, this immediately implies that every peculiar graph is P_{19}-free. A more careful analysis of the definition of a peculiar graph yields the following result.

Lemma 6.24. *Every peculiar graph is P_6-free but not P_5-free.*

Proof. Let H be a peculiar graph, and let A_i, B_i, D_i $(i = 1, 2, 3)$ be a partition of $V(H)$ as mentioned in Definition 6.6. The set $V(H)$ can be partitioned into three cliques, namely $X_1 := A_2 \cup B_1 \cup B_2 \cup D_3$, $X_2 := D_1$ and $X_3 := A_1 \cup A_3 \cup B_3 \cup D_2$. This immediately implies that H is P_7-free, as any

induced path in H contains at most two vertices of any clique. The P_6-freeness of H follows from the observation that for every pair $x, y \in X_2$ we have $N_H(x) \setminus \{y\} = N_H(y) \setminus \{x\}$, which implies that any induced path in H containing vertices of $X_1 \cup X_3$ can only contain at most one vertex from X_2.

Let $a_2 \in A_2$ and $b_3 \in B_3$ be a pair of non-adjacent vertices of H; note that such a pair exists by Definition 6.6. Since none of the sets D_1, D_2, D_3 is empty, H contains an induced path $d_3a_2d_1b_3d_2$, where $d_i \in D_i$, $i = 1, 2, 3$ (see also Figure 6.2). □

The observation that any induced path from s to t in a P_6-free graph H contains at most three other vertices of H immediately implies the following result.

Lemma 6.25. *Both the* Shortest Odd Induced Path *problem and the* Shortest Even Induced Path *problem can be solved in $\mathcal{O}(n^3)$ time for the class of P_6-free graphs.*

Lemma 6.24 and Lemma 6.25 together immediately yield the following.

Corollary 6.26. *Both the* Shortest Odd Induced Path *problem and the* Shortest Even Induced Path *problem can be solved in $\mathcal{O}(n^3)$ time for the class of peculiar graphs.*

6.4.2 A closer look at Tarjan's decomposition algorithm

We now take a closer look at Tarjan's [254] decomposition algorithm, mentioned in Section 6.2. We prove some properties of the clique separator decomposition obtained by this algorithm, and use those properties in the proof of Theorem 6.29 below. We first introduce some additional terminology and describe an algorithm, called the Elimination Game, which is used as a subroutine in Tarjan's decomposition algorithm.

Recall that a graph is *chordal* if it does not contain an induced cycle of length at least 4. If a graph G is a subgraph of a chordal graph H, then H is called a *triangulation* of G. A triangulation H of a graph G is called *minimal* if none of the proper subgraphs of H is a triangulation of G. Consider the following algorithm, known as the *Elimination Game*: given a graph $G = (V, E)$ and an ordering $\pi = v_1, \ldots, v_{|V|}$ of the vertices of G, repeatedly choose a vertex v_i with the lowest index, add edges in order to

make the neighborhood of v_i into a clique, and remove v_i from the graph. The output G_π^+ of the Elimination Game is a triangulation of the input graph G, and the set F_π of edges that are added during the Elimination Game are called *fill edges*. Note that $G_\pi^+ = (V, E \cup F_\pi)$. The total number of fill edges depends on the order π in which the vertices are considered. If the number of fill edges is 0, then the order in which the vertices were considered is called a *perfect elimination ordering* of G. It is well-known that a graph has a perfect elimination ordering if and only if it is chordal [121]. An ordering π is called a *minimal elimination ordering* if G_π^+ is a minimal triangulation of G.

Tarjan's clique separator decomposition algorithm takes as input a connected graph G, and starts by finding a minimal elimination ordering π of the vertices of G. The algorithm then calculates G_π^+ by running the Elimination Game on G and π. For each vertex v of G, the algorithm then computes $C(v) := \{w \mid \pi(v) > \pi(w) \text{ and } vw \in E \cup F_\pi\}$, i.e., the set of neighbors of v in the graph G_π^+ that appear after v in the ordering π, where π is interpreted as a bijection from V to $\{1, \ldots, |V|\}$. The algorithm repeats the following *decomposition step* for each vertex v in increasing order with respect to π: let A be the vertex set of the connected component of $G[V - C(v)]$ containing v, and let $B := V - (C(v) \cup A)$; if $C(v)$ is a clique of G and $B \neq \emptyset$, decompose G into $G' := G[A \cup C(v)]$ and $G'' := G[B \cup C(v)]$; replace G by G''.

A decomposition step is called *successful* if a clique separator is found. Note that such a clique separator is a clique separator of a subgraph of G, as the size of the graph under consideration decreases with every successful decomposition step. However, a simple lemma of Gavril [131] implies that every clique separator found in a successful decomposition step is also a clique separator of the original input graph G. We point out that in every successful decomposition step the graph $G' := G[A \cup C(v)]$ is an atom of G, i.e., does not contain a clique separator (see also [254]). In other words, in every successful decomposition step, a new atom of G is obtained.

When Tarjan's algorithm is run on a graph that has been preprocessed in the way described in Section 6.3.1, i.e., on a graph G that is clean for two simplicial vertices s and t, then we can prove another property of a clique separator found in a successful decomposition step.

Observation 6.27. *Let G be a graph that is clean for two simplicial vertices s and t. Suppose $C(v)$ is a clique separator of a subgraph G'' of G found in a*

successful decomposition step of Tarjan's algorithm. Then $G - C(v)$ consists of exactly two components, one containing s and the other containing t.

Proof. As we mentioned before, $C(v)$ is a clique separator of G as a result of a lemma by Gavril [131], so $G - C(v)$ consists of at least two components. We first show that s and t cannot belong to the same component. Suppose, for contradiction, that s and t belong to the same component D of $G - C(v)$. Let D' be another component of $G - C(v)$, and let $d' \in D'$. Suppose there exists an induced path P from s to t containing d'. Since $C(v)$ is a clique separator of $G - C(v)$ and d' is not in the component of $G - C(v)$ containing both s and t, both the path $s\overrightarrow{P}d'$ and the path $d'\overrightarrow{P}t$ must contain a vertex of $C(v)$. But then P is not an induced path, since $C(v)$ is a clique of G. This contradiction shows that d' is not contained in any induced path from s to t. By definition, this means that d' is irrelevant, contradicting the assumption that G is clean. Hence s and t must belong to two different components D_1 and D_2 of $G - C(v)$, respectively.

Suppose $G - C(v)$ has another component D_3, and let d be a vertex of D_3. Since s, t and d are contained in three different components of $G - C(v)$ and $C(v)$ is a clique of G, there does not exist any induced path from s to t containing d. This means that d is irrelevant, contradicting the assumption that G is clean. We conclude that $G - C(v)$ consists of exactly two components, one containing s and the other containing t. □

We now describe an algorithm that allows us to define an ordering on the atoms of G.

Let G be a graph that is clean for two simplicial vertices s and t. Let $C(v)$ be the clique separator found in a successful decomposition step, and let $G' := G[A \cup C(v)]$ be the corresponding atom of G produced in that step, where A is the vertex set of the connected component of $G[V - C(v)]$ containing v. We define $G_i := G'$, where the index $i \in \{1, \ldots, n\}$ is determined as follows. By Observation 6.27, $G - C(v)$ contains exactly two components D_1 and D_2, where $s \in V(D_1)$ and $t \in V(D_2)$. Note that $v \in A$ and $v \notin C(v)$, which means that v belongs to either D_1 or D_2. If v belongs to D_1, then we choose i to be the smallest integer from $\{1, \ldots, n\}$ that has not yet been used. Otherwise we choose i to be the largest integer from $\{1, \ldots, n\}$ that has not yet been used. We repeat this procedure for each atom G' created in

a successful decomposition step. The graph G'' in the last successful decomposition step is an atom of G, and we define $G_i := G''$, where i is the smallest integer from $\{1, \ldots, n\}$ that has not yet been used. Note that this yields a clique separator decomposition $\mathcal{C}$ of G, where $\mathcal{C} := \{G_1, \ldots, G_k, G_\ell, \ldots, G_n\}$ for some $k < \ell$. For convenience, we relabel the atoms in such a way that the atoms have consecutive indices, i.e., such that $\mathcal{C} := \{G_1, \ldots, G_p\}$, where $p = k + n - \ell + 1$. We point out that s belongs to G_1 and t belongs to G_p as a result of Observation 6.27.

Before we prove the main result of this section, we prove one more useful lemma.

Lemma 6.28. *Let G be a graph that is clean for two simplicial vertices s and t. Let $\mathcal{C} := \{G_1, \ldots, G_p\}$ be a clique separator decomposition of G obtained by Tarjan's decomposition algorithm, where the indices of the atoms have been determined using the above procedure. Every induced path in G from s to t passes through each of the atoms $G_1, \ldots, G_p$, and passes through them in increasing order, i.e., passes through G_i before passing through G_j for every $1 \leq i < j \leq p$.*

Proof. Let P be an induced path in G from s to t. We first show that P contains a vertex of every atom of $\mathcal{C}$. Suppose, for contradiction, that P does not contain any vertex of some atom $G_i \in \mathcal{C}$. Let C be the clique separator that was found in the successful decomposition step that produced atom G_i. By Observation 6.27, the graph $G - C$ consists of two components D_1 and D_2, where $s \in V(D_1)$ and $t \in V(D_2)$. Note that P does not contain a vertex of C, since $C \subseteq V(G_i)$. Hence the graph $G - C$ contains the path P from s to t. This contradicts the fact that s and t belong to different components of $G - C$. We conclude that P must contain a vertex of every atom of $\mathcal{C}$.

Since $s \in V(G_1)$, P passes through atom G_1 first. Suppose P does not visit the atoms of $\mathcal{C}$ in the order $G_1, G_2, \ldots, G_p$, and let G_i be the first atom that P "skips". Let G_j be the first atom that P visits after leaving atom G_{i-1}, $j > i$. Let C be the clique separator that was found in the successful decomposition step that produced G_i. By Observation 6.27, the graph $G - C$ consists of two components D_1 and D_2, where $s \in V(D_1)$ and $t \in V(D_2)$. From the description of the algorithm used to determine the indices of the atoms of $\mathcal{C}$, it follows that $V(G_j) \setminus C \subseteq V(D_1)$ would have implied $j < i$; a

contradiction. Similarly, $V(G_{i-1}) \setminus C \subseteq V(D_2)$ would have implied $i < i-1$; a contradiction. Hence $V(G_{i-1}) \setminus C \subseteq V(D_1)$ and $V(G_j) \setminus C \subseteq V(D_2)$. In particular, we have $s' \in V(D_1)$ and $t' \in V(D_2)$, where $s' \in V(G_{i-1})$ is the last vertex of G_{i-1} that P visits before visiting G_j, and $t' \in V(G_j)$ is the first atom that P visits after leaving G_{i-1}. Note that s' and t' are adjacent vertices of P, and that neither s' nor t' is contained in C since $C \subseteq V(G_i)$. But then the graph $G - C$ contains a path from s to t, using the edge $s't'$. This contradiction to the fact that s and t belong to different components of $G - C$ finishes the proof of Lemma 6.28. □

6.4.3 Shortest paths in claw-free perfect graphs

We are now ready to prove the main result of this section.

Theorem 6.29. *Both the* Shortest Odd Induced Path *problem and the* Shortest Even Induced Path *problem can be solved in* $\mathcal{O}(n^7)$ *time for the class of claw-free perfect graphs.*

Proof. In order to prove Theorem 6.29, we present an algorithm that solves both the Shortest Odd Induced Path problem and the Shortest Even Induced Path problem on claw-free graphs in $\mathcal{O}(n^7)$ time. The algorithm takes as input a claw-free perfect graph and two of its vertices s and t. We first preprocess this input graph by performing the two steps of the preprocessing procedure described in Section 6.3.1. This way we obtain a claw-free graph G, such that s and t are simplicial vertices of G, and G is clean for s and t. The preprocessing phase can be done in $\mathcal{O}(n^5)$ time by Lemma 6.14. As a result of Lemma 6.14, in order to prove Theorem 6.29 it suffices to show that we can solve the two problems on G in $\mathcal{O}(n^7)$ time.

Next we find a clique separator decomposition of G by using Tarjan's decomposition algorithm described in Section 6.4.2. Let $\mathcal{C} := \{G_1, \ldots, G_p\}$ be a clique separator decomposition of G obtained by Tarjan's decomposition algorithm, where the indices of the atoms have been determined using the procedure described just before Lemma 6.28. We can find such a clique separator decomposition in $\mathcal{O}(n^3)$ time by Theorem 6.2. Let $X_{i,i+1} := V(G_i) \cap V(G_{i+1})$ and let $\mathcal{X}$ be the set containing all the sets $X_{i,i+1}$. It is clear from the description of Tarjan's algorithm in Section 6.4.2 that every set in $\mathcal{X}$ is a non-empty clique separator of G. As we pointed out just before Lemma 6.28, s belongs

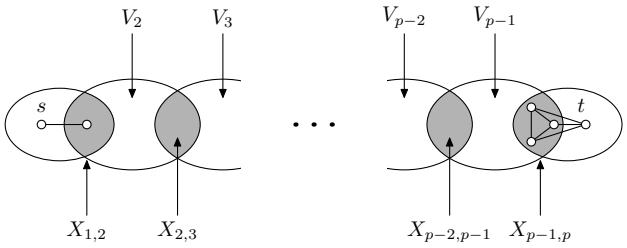

Figure 6.6: Structure of the graph G with respect to the clique separator decomposition $\mathcal{C}$.

to G_1 and t belongs to G_p. Since G does not contain irrelevant vertices, we have $s \in V(G_1) \setminus X_{1,2}$ and $t \in V(G_p) \setminus X_{p-1,p}$.

Note that, in general, a set $X_{i-1,i}$ might share vertices with the set $X_{i,i+1}$, and possibly with other sets in $\mathcal{X}$. Let us for the moment assume that this is not the case, i.e., that the sets in $\mathcal{X}$ are pairwise disjoint. At the very end of this proof we will explain why we can make this assumption without loss of generality. Define $X_{0,1} := \emptyset$, $X_{p,p+1} := \emptyset$ and $V_i := V(G_i) \setminus (X_{i-1,i} \cup X_{i,i+1})$ for $i = 1, \ldots, p$. Let $W_i := V_1 \cup \cdots \cup V_i \cup X_{1,2} \cup \cdots \cup X_{i-1,i} = V(G_1) \cup \ldots \cup V(G_{i-1}) \cup (V(G_i) \setminus X_{i,i+1})$ for $i = 1, \ldots, p-1$. See Figure 6.6 for a schematic representation of graph G with respect to the clique separator decomposition $\mathcal{C}$.

Claim 1. *Let $G_i \in \mathcal{C}$. We can solve* Shortest Odd Induced Path *and* Shortest Even Induced Path *in $\mathcal{O}(n^4)$ time for any induced subgraph of G_i.*

Let G' be an induced subgraph of one of the atoms $G_i \in \mathcal{C}$. The graph G_i is a claw-free perfect graph without a clique separator, so G_i is either elementary or peculiar by Theorem 6.8. By Lemma 6.5 and Lemma 6.7 we can decide in $\mathcal{O}(n^3)$ time whether G_i is elementary or peculiar. If G_i is peculiar, then G_i is P_6-free by Lemma 6.24. Since every induced subgraph of an elementary (respectively P_6-free) graph is elementary (respectively P_6-free), we can solve Shortest Odd Induced Path and Shortest Even Induced Path for the graph G' in $\mathcal{O}(n^4)$ time as a result of Lemma 6.23 and Lemma 6.25, respectively. This finishes the proof of Claim 1.

We observe that any induced path from s to t contains either one vertex or

two vertices of each $X_{i,i+1}$, since each set $X_{i,i+1}$ is a clique. We now restrict our attention to the graph G_1. We claim that all vertices of G_1 belong to the closed neighborhood of s. Suppose there is a vertex $v \in V(G_1) \setminus N[s]$. Then $N(s)$ is a clique separator of G_1, contradicting the assumption that G_1 is an atom. Hence we know that for every vertex $x \in X_{1,2}$, there exists only one induced path from s to x, and it has length 1.

In order to find a shortest odd and a shortest even induced path from s to any vertex in $X_{p-1,p}$, we run the following algorithm for increasing $i = 2, \ldots, p-1$. For each vertex v in $X_{i,i+1}$ we perform the following two steps. We state the two steps first, before we describe how to perform them below.

Step 1. Find a shortest odd induced path from s to v in $G[W_i \cup \{v\}]$, or conclude that such a path does not exist, and find a shortest even induced path from s to v in $G[W_i \cup \{v\}]$, or conclude that such a path does not exist.

Step 2. For each $v' \in X_{i,i+1} \setminus \{v\}$, find a shortest odd induced path from s to v in $G[W_i \cup \{v, v'\}]$ using edge $v'v$, or conclude that such a path does not exist, and find a shortest even induced path from s to v in $G[W_i \cup \{v, v'\}]$ using edge $v'v$, or conclude that such a path does not exist.

To execute Step 1, we act as follows. For all $u \in X_{i-1,i}$, we find a shortest odd (even) induced path from u to v in the graph $G' := G[V_i \cup \{u, v\}]$, or conclude that such a path does not exist. Since G' is an induced subgraph of G_i, we can find a shortest odd (even) induced path from u to v in G' in $\mathcal{O}(n^4)$ time as a result of Claim 1. Combining those shortest induced paths of both parities with the shortest induced paths of both parities from s to u in $G[W_{i-1} \cup \{u\}]$ yields at most four induced paths from s to v in $G[W_i \cup \{v\}]$. To check whether there exists a shorter odd or even induced path from s to v, using *two* vertices of $X_{i-1,i}$, we act as follows. For each $u' \in X_{i-1,i} \setminus \{u\}$, we find a shortest odd (even) induced path from u to v in the graph $G[(V_i \setminus N_G(u')) \cup \{u, v\}]$. We combine those paths of both parities with the shortest induced paths of both parities from s to u, using edge $u'u$, in the graph $G[W_{i-1} \cup \{u, u'\}]$. This way we are guaranteed to find both a shortest odd and a shortest even induced path from s to v in $G[W_i \cup \{v\}]$, unless one of those paths does not exist. For step 2 we perform similar checks in $\mathcal{O}(n^4)$ time.

After we have completed both steps for $i = p - 1$, we have found (if they exist) shortest odd and shortest even induced paths from s to every vertex in $X_{p-1,p}$, both paths using one and paths using two vertices of $X_{p-1,p}$. Recall that $V(G_1) \subseteq N[s]$. Similarly, we have $V(G_p) \subseteq N[t]$, which means that there is exactly one induced path from any vertex in $X_{p-1,p}$ to t, and it has length 1. This way we find a shortest odd and a shortest even induced path from s to t, or conclude that such a path does not exist. Since $\sum_{i=1}^{p-1} |X_{i,i+1}| \leq n$, we only have to perform the $\mathcal{O}(n^4)$ procedure for finding shortest induced paths on $\mathcal{O}(n^3)$ induced subgraphs of G. Hence the overall time complexity is $\mathcal{O}(n^7)$.

What remains is to argue why we may assume that the sets in $\mathcal{X}$ are pairwise disjoint. Suppose that the algorithm has processed atoms $G_1, \ldots, G_{i-1}$, and is about to process atom G_i. The algorithm has found the shortest odd (even) induced paths from s to every vertex in $X_{i-1,i}$, using either one or two vertices of $X_{i-1,i}$. After processing atom G_i, the algorithm has extended those paths to shortest odd (even) induced paths from s to every vertex in $X_{i,i+1}$, using either one or two vertices of $X_{i,i+1}$. Suppose $X_{i-1,i}$ and $X_{i,i+1}$ overlap, and let x be in $X_{i-1,i} \cap X_{i,i+1}$. As a result of Lemma 6.28 and the observation that any induced path can pass through a clique only once, the paths found for x just before the algorithm starts processing atom G_i are exactly the same as the paths found after G_i is processed. In other words, we do not have to perform Steps 1 and 2 for any vertex in $X_{i-1,i} \cap X_{i,i+1}$, since it will make no difference to the final solution. That means we can redefine $X_{i,i+1}$ as follows: $X_{i,i+1} := X_{i,i+1} \setminus X_{i-1,i}$. Similar arguments imply that the same holds for any other set in $X_{j,j+1} \in \mathcal{X}$ with $j > i$ that overlaps with $X_{i-1,i}$. Hence we may assume that the sets in $\mathcal{X}$ are pairwise disjoint, and the clique separator decomposition of G looks like the one sketched in Figure 6.6. □

6.5 Conclusion

We have proved that both the Odd Induced Path problem and the Even Induced Path problem, and consequently the Parity Path problem, can be solved in $\mathcal{O}(n^5)$ time for the class of claw-free graphs. This immediately implies that we can also decide in polynomial time whether a claw-free graph

contains an odd induced path between *every* pair of vertices. We also showed how we can find a *shortest* induced path of given parity between two specified vertices of a claw-free perfect graph in $\mathcal{O}(n^7)$ time. Does there exist a polynomial time algorithm for the Shortest Odd Induced Path and Shortest Even Induced Path problems for *general* claw-free graphs? Another interesting question is whether or not there exists a polynomial time algorithm for the Odd Induced Path and Even Induced Path problems for the class of planar graphs.

One of Bienstock's [32] NP-complete problems is to decide whether a graph contains an odd hole passing through a given vertex. We showed that this problem, as well as the variant where we want to find an even hole through a given vertex, can be solved in $\mathcal{O}(n^7)$ time for the class of claw-free graphs. Very recently, Shrem et al. [250] obtained a polynomial time algorithm for detecting a shortest odd hole in a claw-free graph. There are a number of problems in the literature related to finding holes in graphs. Checking if a graph has no hole is equivalent to deciding if the graph is chordal. It is well-known that this problem can be solved in linear time [236]. An interesting related problem is to decide if a graph has an *odd* hole. The computational complexity of this problem remains open, even though a seemingly similar problem —deciding if a graph has an *even* hole— can be solved in polynomial time [63].

Chapter 7

Finding longest cycles in claw-free graphs

A preliminary version of this chapter, dealing with finding hamiltonian cycles in claw-free graphs rather than longest cycles, has appeared in the following paper.

[50] H.J. Broersma, F.V. Fomin, P. van 't Hof, and D. Paulusma. Fast exact algorithms for hamiltonicity in claw-free graphs. In: *Proceedings of the 35th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2009)*, volume 5911 of *Lecture Notes in Computer Science*, pages 44–53, Springer, 2009.

After looking for induced paths in the previous chapter, we turn our attention to cycles in this chapter. The Hamiltonian Cycle problem, asking whether or not a graph contains a cycle passing through all its vertices, is a well-known NP-complete problem. In fact, it was one of the first problems shown to be NP-complete, as Karp included the problem in the list of 21 assorted combinatorial and graph theoretical problems that he showed to be NP-complete in his 1972 landmark paper [176], only a year after Cook [78] introduced the concept and identified Satisfiability as the first NP-complete problem. In this chapter we study the problem of finding a longest cycle in a graph, which is a more general, and therefore harder, problem than the Hamiltonian Cycle problem. Like we did in Chapter 6 we restrict the problem to the class of claw-free graphs. However, unlike the problems stud-

ied in the previous chapter, the problem of finding a longest cycle remains NP-hard under this restriction. We present two exact exponential time algorithms that solve this problem significantly faster than a trivial algorithm, one using exponential space and one using polynomial space. At the heart of our algorithms lies a beautiful structural closure concept for claw-free graphs developed by Ryjáček [238]. We also prove some structural results about closed trails in a graph, and use those results to obtain fast exact algorithms.

7.1 Background and results

A *longest* cycle in a graph G is a cycle in G that has the largest number of vertices among all cycles in G. In this chapter we study the following problem.

Longest Cycle
Instance: A graph G.
Task: Find a longest cycle in G.

This problem generalizes the well-known NP-complete decision problem Hamiltonian Cycle (cf. [127]) that asks whether a graph G has a *hamiltonian cycle*, i.e., a cycle passing through all vertices of G. The Hamiltonian Cycle problem can be seen as a special case of the well-known Traveling Salesman problem. The input of the latter problem is a complete graph together with an edge weighting. The goal is to find a hamiltonian cycle of minimum total weight. Held and Karp [151] present a classic dynamic programming algorithm that solves the Traveling Salesman problem in $\mathcal{O}^*(2^n)$ time and $\mathcal{O}^*(2^n)$ space for graphs on n vertices. Polynomial space algorithms for the Hamiltonian Cycle problem were rediscovered several times [18, 178, 182]. It is a major and long outstanding open problem whether the Hamiltonian Cycle problem, and more generally the Traveling Salesman problem, can be solved in $\mathcal{O}^*(c^n)$ time for some constant $c < 2$.

For some graph classes for which the Hamiltonian Cycle, and consequently the Traveling Salesman problem, remains NP-complete, faster exact algorithms have been designed. For planar graphs, and more generally for graphs excluding some fixed graph as a minor, the Hamiltonian Cycle

problem can be solved in $\mathcal{O}^*(c^{\sqrt{n}})$ for some constant c (cf. [93, 94, 265]). The Traveling Salesman problem can be solved in $\mathcal{O}^*(1.251^n)$ time for cubic graphs [170] and in $\mathcal{O}^*(1.890^n)$ time for graphs with maximum degree 4 [98]. Both algorithms use polynomial space. For graphs with maximum degree 4, an algorithm with time complexity $\mathcal{O}^*(1.733^n)$ is known [132], but this algorithm uses exponential space. More generally, Björklund et al. [33] present an algorithm that solves the Traveling Salesman problem in $\mathcal{O}^*((2-\epsilon)^n)$ for graphs with bounded degree, where $\epsilon > 0$ only depends on the maximum degree but not on the number of vertices. They show that this bound can be improved further for regular triangle-free graphs. These algorithms use exponential space. They also present a $\mathcal{O}^*((2-\epsilon)^n)$ time algorithm that uses polynomial space for bounded degree graphs in which the edges have bounded integer weights.

Just as we did in Chapter 6, we restrict our attention in this chapter to the class of claw-free graphs. The Hamiltonian Cycle problem is NP-complete for claw-free graphs; in fact, the problem remains NP-complete even on 3-connected cubic planar claw-free graphs [195]. This immediately implies that the Longest Cycle problem is NP-hard for claw-free graphs as well. Although this implies that it is unlikely that the problem can be solved in polynomial time, we present in this chapter two exact algorithms that solve the Longest Cycle problem on claw-free graphs significantly faster than in time $\mathcal{O}^*(2^n)$: our first algorithm uses $\mathcal{O}^*(1.6818^n)$ time and exponential space, and our second algorithm uses $\mathcal{O}^*(1.8878^n)$ time and polynomial space. We give an outline of our algorithms in Section 7.3, where we show that the problem of finding a longest cycle in a claw-free graph can be translated into the problem of finding an optimum closed trail (i.e., a closed trail dominating the largest number of edges) of an associated graph. Two exact algorithms for finding such an optimum closed trail are presented in Section 7.4. We start in Section 7.2 by proving several structural properties of closed trails. These results will be exploited in the subsequent sections to guarantee the time complexities of our algorithms. We conclude this section by introducing some additional terminology and notation that will be used throughout this chapter.

Recall that we defined a graph to have a non-empty vertex set. Hence, if we say that a particular graph "exists" we mean that its vertex set is non-

empty. The maximum degree among the vertices of a graph G is denoted by $\Delta(G)$. A graph is called *even* if all its vertices have even degree. A graph is called a *closed trail* if it is a connected even graph. Note that an even graph, and a closed trail in particular, might consist of a single vertex. A closed trail that does not consist of a single vertex is called *non-trivial*; note that a non-trivial closed trail contains at least three vertices. Let T be a closed trail of a graph H. An edge $e \in E(H)$ is *dominated by* T if T contains at least one of the end vertices of e. In this context "dominated" means "edge-dominated", and this is the case whenever we speak of domination in this chapter. Note that, by definition, every edge of a non-trivial closed trail T is dominated by T itself. For any closed trail T of H, we denote by $\beta(T)$ the number of edges of H dominated by T, i.e., $\beta(T) := |E(H) \setminus E(H - V(T))|$. If every edge of H is dominated by T, i.e., if $\beta(T) = |E(H)|$, then we say that T is a *dominating closed trail* of H. An *optimum non-trivial closed trail* or ONCT of H is a non-trivial closed trail of H that dominates at least as many edges of H as any other non-trivial closed trail of H. A closed trail T of a graph H is called an *optimum closed trail* or OCT if $\beta(T) \geq \beta(T')$ for any closed trail T' of H. Note that every graph has an OCT, and that an OCT of H is either an ONCT of H, or a single vertex with degree $\Delta(H)$ in case $\Delta(H) > \beta(T)$ for any non-trivial closed trail T of H.

For any integer $k \geq 1$, a graph H is called *k-degenerate* if every subgraph of H (including H itself) has a vertex with degree at most k. We say that H is *k-ordered* if H allows a vertex ordering $v_1, \ldots, v_{|V(H)|}$ such that, for $1 \leq i \leq |V(H)|$, the graph $H[\{v_1, \ldots, v_i\}]$ is connected and v_i has at most k neighbors in $H[\{v_1 \ldots, v_i\}]$.

7.2 Closed trails of low degeneracy and ordering

In this section we study structural properties of closed trails. We will use such properties in the exact algorithms for finding an optimum closed trail presented in Section 7.4.

A cycle C of a connected graph H is called *removable* if the graph $H - E(C)$ is connected, and *non-separating* if $H - V(C)$ is connected. The following result is due to Thomassen and Toft [257].

Theorem 7.1 ([257]). *Every connected graph with minimum degree at least 3 has an induced non-separating cycle.*

Theorem 7.1 implies the following result.

Corollary 7.2. *Every connected graph with minimum degree at least 3 has an induced removable cycle.*

Proof. Let H be a connected graph with minimum degree at least 3. By Theorem 7.1, H has an induced non-separating cycle C. Since $H - V(C)$ is connected, all vertices of $V(H) \setminus V(C)$ belong to the same component of $H - E(C)$. Since H has minimum degree at least 3 and C is an induced cycle, every vertex of C has a neighbor in $V(H) \setminus V(C)$. Hence $H - E(C)$ is connected, so C is removable. □

Using Corollary 7.2, we can prove the following result.

Lemma 7.3. *Every closed trail contains a 2-degenerate spanning closed trail.*

Proof. Since a closed trail consisting of a single vertex is 2-degenerate, the lemma holds for such closed trails. We claim that every non-trivial closed trail contains a 2-degenerate spanning closed trail. Let H be a counterexample to this claim with $|E(H)|$ minimum, i.e., H is a non-trivial closed trail which does not contain a 2-degenerate spanning closed trail. In particular, H itself is not 2-degenerate. We repeatedly remove vertices from H with degree at most 2 in the current subgraph of H as long as possible. Let H' be the subgraph of H we obtain this way. Since H is not 2-degenerate, H' indeed exists. Let H_1 be a component of H'. Since H' has minimum degree at least 3, H_1 has a removable cycle C by Corollary 7.2. Then C is also a removable cycle in H, since H is a connected supergraph of H_1. Hence the graph $H - E(C)$ is a spanning non-trivial closed trail of H. Since H is a counterexample, $H - E(C)$ is not 2-degenerate and $H - E(C)$ does not contain a 2-degenerate spanning closed trail. But then $H - E(C)$ is a counterexample to the claim that every non-trivial closed trail contains a 2-degenerate spanning closed trail, contradicting the minimality of H. □

The next lemma shows that the notions of degeneracy and ordering are closely related.

Lemma 7.4. *Every connected k-degenerate graph is $(k+1)$-ordered, for any $k \geq 1$.*

Proof. Let H be a connected k-degenerate graph, and suppose for contradiction that H is not $(k+1)$-ordered. We repeatedly remove vertices from H with degree at most $k+1$ in the current subgraph of H, until we cannot remove any vertex with degree at most $k+1$ without making the current subgraph disconnected. Let H' be the resulting (connected) subgraph of H. Since H is not $(k+1)$-ordered, H' indeed exists. Let U consist of all vertices with degree at most k in H'. By our procedure, every vertex of U is a cut vertex of H', and since H is k-degenerate, U is not empty. Hence H' contains at least one cut vertex. Let D be an end-block of H', i.e., a maximal 2-connected subgraph of H' containing exactly one cut vertex x of H'. By our procedure, every vertex of $D - x$ has degree at least $k+2$ in H', which means that every vertex of $D - x$ has degree at least $k+1$ in $D - x$. Since $D - x$ is a subgraph of H, this contradicts the k-degeneracy of H. □

It is well-known that a connected graph is 1-degenerate if and only if it is a tree. It is not hard to see that every tree, and therefore every connected 1-degenerate graph, is 1-ordered. This means that Lemma 7.4 can be slightly strengthened for $k = 1$. The following result shows that Lemma 7.4 is best possible for $k \geq 2$.

Proposition 7.5. *For any $k \geq 2$, there exists a connected k-degenerate graph that is not k-ordered.*

Proof. For any $k \geq 2$, let G_k be the graph constructed as follows. Start with the join of C_{k+2} and $\overline{K_{k-1}}$, i.e., the graph obtained from the disjoint union of a cycle of length $k+2$ and an independent set S on $k-1$ vertices by making every vertex of the cycle adjacent to every vertex of S. Let H be the graph obtained from this graph by removing one edge cw, where c is a vertex of the cycle and w is a vertex of S. Take k copies $H_1, \ldots, H_k$ of the graph H, and let $c_1, \ldots, c_k$ denote the copies of vertex c in $H_1, \ldots, H_k$, respectively. Finally, G_k is obtained by adding a vertex x and edges $xc_1, \ldots, xc_k$. As an example, the graph G_3 is depicted in Figure 7.1.

It is straightforward to verify that G_k is k-degenerate. We claim that G_k is not k-ordered. For contradiction, suppose G_k is k-ordered. By definition, G_k

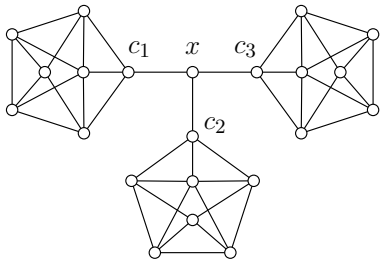

Figure 7.1: The graph G_3, which is 3-degenerate but not 3-ordered.

has an ordering $v_1, \ldots, v_{|V(G_k)|}$ of its vertices such that, for $1 \leq i \leq |V(G_k)|$, the graph $G_k[\{v_1, \ldots, v_i\}]$ is connected and v_i has at most k neighbors in $G_k[\{v_1 \ldots, v_i\}]$. Since x is the only vertex of G_k with degree at most k in G_k, $x = v_{|V(G_k)|}$. But the fact that x is a cut vertex of G_k implies that the graph $G_k[\{v_1, \ldots, v_i\}]$ is not connected for $i = |V(G_k)| - 1$, yielding the desired contradiction. □

Lemma 7.3 and Lemma 7.4 together imply the following result, which will be used in the exact algorithms described in the next two sections.

Corollary 7.6. *Every graph has a 2-degenerate 3-ordered optimum closed trail.*

Proof. Let T be an optimum closed trail of a graph H, and let $S \subseteq E(H)$ be the set of edges of H that are dominated by T. By Lemma 7.3, the graph T contains a 2-degenerate spanning closed trail T'. Since $V(T') = V(T)$, the set of edges of H dominated by T' is exactly the set S. Hence T' is an optimum closed trail of H. Since T' is 2-degenerate, T' is 3-ordered as a result of Lemma 7.4. □

7.3 Two exact algorithms for finding a longest cycle

In this section we explain our two algorithms that solve the Longest Cycle problem for a claw-free graph G on n vertices. We assume from now on that G is connected, since we can treat the components of G separately in case G is disconnected. We also assume that G contains a longest cycle, i.e., that

G is not a tree. Note that we can check in polynomial time whether G is a connected graph other than a tree.

For the first, third and fourth step below we do not have to develop any new theory or algorithms, but can rely on the beautiful existing machinery from the literature.

Step 1: restrict to the preimage graph H of the closure of G

Since G is claw-free, the set of neighbors of each vertex v in G induces a subgraph with at most two components. If this subgraph has two components, both of them must be cliques. If the subgraph induced by $N_G(v)$ is connected but not complete, we can perform an operation called *local completion of G at v* by adding edges joining all pairs of non-adjacent vertices in $N_G(v)$. We recursively repeat the local completion operation, as long as this is possible. This way we obtain the *closure* $cl(G)$ of G.

Ryjáček [238] showed that the closure of G is uniquely determined, i.e., that the ordering in which one performs the local completions does not matter. This means we can obtain $cl(G)$ in polynomial time. Ryjáček [238] also showed that the length of a longest cycle in G equals the length of a longest cycle in $cl(G)$. In particular, G is hamiltonian if and only if $cl(G)$ is hamiltonian. Furthermore he showed that for any claw-free graph G there is a unique (triangle-free) graph H such that $L(H) = cl(G)$. We can obtain the preimage graph of a line graph in polynomial time (see e.g. [237]). Hence we can compute the unique graph H with $L(H) = cl(G)$ in polynomial time.

Step 2: find an OCT of H

Harary and Nash-Williams [146] showed that a hamiltonian cycle in a line graph of any connected graph on at least three vertices corresponds to a dominating closed trail of the graph itself. By an easy variation on their arguments, many researchers have shown that a longest cycle in such a line graph corresponds to an optimum closed trail of the graph itself. This result, combined with the results from the previous step, implies that finding a longest cycle in G corresponds to finding an OCT of H. In Section 7.4 we present two exact algorithms for finding an OCT of a connected graph with n edges: one algorithm that uses $\mathcal{O}^*(1.6818^n)$ time and exponential space, and one algorithm that uses $\mathcal{O}^*(1.8878^n)$ time and polynomial space.

Step 3: translate the OCT of H back into a longest cycle in $cl(G)$

Let T be the OCT of H that we obtained in Step 2. We construct a longest cycle in $cl(G)$ by traversing T, picking up the edges (corresponding to vertices in $cl(G)$) one by one and inserting dominated edges as soon as an end vertex of a dominated edge is encountered. For traversing T we use the polynomial time algorithm that finds a eulerian tour in a connected even graph (cf. [91]). We point out that in case T consists of a single vertex v, a longest cycle in $cl(G)$ is any cycle spanning the clique in $cl(G)$ that corresponds to all edges of H dominated by v.

Step 4: translate the longest cycle in $cl(G)$ into one in G

We can translate the longest cycle in $cl(G)$ obtained in Step 3 into a longest cycle in G in polynomial time by using the method described by Broersma and Paulusma [51], who show how to translate a 2-factor of $cl(G)$ into a 2-factor of G. (A *2-factor* of G is a spanning subgraph of G in which all vertices have degree 2.) It is straightforward to adapt this process and apply it to a single cycle D in $cl(G)$ such that we find, in polynomial time, a cycle C in G with the same length as D.

We mentioned that the first, third and fourth step above can be performed in polynomial time. We also mentioned that we will show in Section 7.4 that the second step can be executed in $\mathcal{O}^*(1.6818^n)$ time using exponential space, or in $\mathcal{O}^*(1.8878^n)$ time using polynomial space. Hence, we have found the following.

Theorem 7.7. *The* Longest Cycle *problem, and consequently the* Hamiltonian Cycle *problem, for a claw-free graph on n vertices can be solved in $\mathcal{O}^*(1.6818^n)$ time, using exponential space. The two problems can also be solved in $\mathcal{O}^*(1.8878^n)$ time, using polynomial space.*

7.4 Two exact algorithms for finding an OCT

In this section we present two exact algorithms for solving the following problem.

Optimum Closed Trail (OCT)
Instance: a connected graph H.
Task: find an optimum closed trail of H.

Both algorithms can be outlined as follows.

Algorithm solving the Optimum Closed Trail **problem**

Input : a connected graph H
Output : an optimum closed trail of H

Test whether or not H is a tree

If H is a tree, output a vertex v of H with degree $\Delta(H)$
If H is not a tree, find an optimum non-trivial closed trail T of H

Test whether or not $\beta(T) \geq \Delta(H)$

If $\beta(T) \geq \Delta(H)$, output T
If $\beta(T) < \Delta(H)$, output a vertex v of H with degree $\Delta(H)$

The difference between the two algorithms is the way in which they compute an ONCT of H in case H is not a tree. To find an ONCT of a connected graph H other than a tree, both algorithms start by branching on vertices of low degree by the same branching procedure, explained in Section 7.4.1. This way both algorithms obtain a set of subproblems. Each subproblem has the original graph H as input. However, for some subset of the edges of H it is already decided whether they will be included in or excluded from the ONCT. Our first algorithm, described in Section 7.4.2, solves each of the subproblems using dynamic programming. Our second algorithm, described in Section 7.4.3, solves each of the subproblems by guessing the remaining edges of a possible ONCT.

7.4.1 Branching on vertices of low degree

Let H be an instance of the OCT problem, and suppose H is not a tree. As can be seen in the outline of the algorithms at the start of Section 7.4, both algorithms find an ONCT of H. In order to find an ONCT of H, both algorithms start by assigning a so-called *parity label* $\ell(v) \in \{0, 1\}$ to each vertex v of H. Note that if T is an ONCT of H, then $d_T(v)$ is even for every $v \in V(H)$. After all, a vertex is either not in T (i.e., $d_T(v) = 0$), or a vertex has an even number of incident edges in T, since T is a non-trivial closed trail. Hence we initially set $\ell(v) := 0$ for every $v \in V(H)$.

Both algorithms now branch on vertices of degree at most d^*, thus creating a number of subproblems; more specifically, $d^* = 4$ for our first algorithm, and $d^* = 12$ for our second algorithm. The choice of these values of d^* is explained in the next sections. During the branching process, the size of the graphs under consideration decreases, and we might change the ℓ-labels of certain vertices.

Suppose v is a vertex of degree $d \leq d^*$ in H. If $\ell(v) = 0$ (respectively $\ell(v) = 1$), then the algorithm branches into 2^{d-1} subproblems, each subproblem corresponding to a possible way of choosing an even (respectively odd) number $0 \leq p \leq d$ of edges incident with v that are guessed to be in the ONCT. We call the chosen edges *old trail edges*. For each choice W of old trail edges, we perform the following two operations:

1. set $\ell(w) := \ell(w) + 1 \pmod 2$ for every w with $vw \in W$;
2. delete v and all its d incident edges.

Repeat this procedure as long as the remaining graph contains a vertex of degree at most d^*. Let H' be the resulting graph. Then H' has minimum degree $d^* + 1$ and each vertex $u \in V(H')$ has some label $\ell(u) \in \{0, 1\}$. Let $E(H) = E(H') \cup R(H') \cup W(H')$, where $W(H')$ contains all old trail edges and $R(H')$ contains all other edges we removed from H. In the next stage, edges in $W(H')$ will be assumed to be in the ONCT we are looking for, whereas edges in $R(H')$ will be assumed not to be in the ONCT. If there exists a vertex $v \in V(H) \setminus V(H')$ incident with an odd number of old trail edges, then we discard the subproblem. The reason for this is the fact that we can never obtain a non-trivial closed trail in such a subproblem, since v will have odd degree in that trail and that is not possible. Otherwise, we keep the subproblem and call the tuple $(H', W(H'), \ell)$ a *stage-2 tuple*.

Lemma 7.8. *The branching phase creates $T(n_1) = \mathcal{O}^*(2^{\frac{d^*-1}{d^*}n_1})$ stage-2 tuples, where n_1 is the total number of edges deleted during this phase.*

Proof. Since for a vertex v of degree d we remove d edges and create 2^{d-1} subgraphs, we find $T(n_1) = 2^{d-1} \cdot T(n_1 - d)$, which yields $T(n_1) = \mathcal{O}^*(2^{\frac{d-1}{d}n_1})$. Since $d \leq d^*$, we end up with $\mathcal{O}^*(2^{\frac{d^*-1}{d^*}n_1})$ stage-2 tuples. □

We point out that the time complexity in Lemma 7.8 is $\mathcal{O}^*(1.6818^{n_1})$ if $d^* = 4$ and $\mathcal{O}^*(1.8878^{n_1})$ if $d^* = 12$.

7.4.2 An $\mathcal{O}^*(1.6818^n)$ time algorithm

In this section, we start by explaining how the first of our two algorithms for the OCT problem finds an ONCT of the input graph H in case H is not a tree. We then prove that our first algorithm for finding an OCT of a connected graph H is correct, and that it runs in $\mathcal{O}^*(1.6818^n)$ time.

Let H be an input of the OCT problem other than a tree. In case H has vertices of degree at most 4, we apply the branching procedure described in Section 7.4.1. Suppose that during the branching process n_1 edges were deleted (possibly $n_1 = 0$). Then, by Lemma 7.8, $\mathcal{O}^*(1.6818^{n_1})$ stage-2 tuples $(H', W(H'), \ell)$ have been created. Each of these stage-2 tuples will be processed using the dynamic programming procedure described below.

Let $(H', W(H'), \ell)$ be a stage-2 tuple. If $W(H')$ forms a dominating closed trail of H, i.e., if every edge of H has at least one end vertex in common with an edge in $W(H')$, then we have found an optimum closed trail and the algorithm outputs this trail. If this is not the case, then we enter the dynamic programming phase. In this phase, we consider each $u \in V(H')$. We define two labelings $\ell^* : \{u\} \to \{0,1\}$ with $\ell^*(u) := \ell(u)$ and $\bar{\ell} : \{u\} \to \{0,1\}$ with $\bar{\ell}(u) := \ell(u) + 1 \pmod 2$. We say that $(\{u\}, \ell^*)$ is an *option* if u is incident with at least one old trail edge. Otherwise $(\{u\}, \ell^*)$ is not an option. Furthermore, for every $u \in V(H')$, $(\{u\}, \bar{\ell})$ is not an option.

Suppose we know for all sets $S \subseteq V(H')$ of size at most k and all labelings $\ell' : S \to \{0,1\}$ whether (S, ℓ') is an option or not. Then for each set $S \subseteq V(H')$ of size k, for each vertex $v \in V(H') \setminus S$, and for each $\{0,1\}$-labeling ℓ' of $S \cup \{v\}$, we do as follows. Let p be the number of old trail edges incident with v. We consider every possible way of choosing $0 \leq q \leq 3$ edges incident with v and a vertex in S. The chosen edges will be referred to as *new trail edges*. For each choice N of new trail edges, we set $\ell'(x) := \ell'(x) + 1 \pmod 2$ for every $x \in S$ with $vx \in N$. We then perform the following three tests.

(1) Check if (S, ℓ') is an option.
(2) Check if $p + q$ is even if $\ell'(v) = 0$ and odd if $\ell'(v) = 1$.
(3) If $q = 0$, check if there is a path from v to S in H only using old trail edges.

Only if the answers to tests (1), (2) and (3) are all affirmative, we say that $(S \cup \{v\}, \ell')$ is an option. If so, we also check whether

(4) for each old trail edge e there is a path, consisting of only old trail edges, connecting e to a vertex in $S \cup \{v\}$;
(5) each vertex x in $S \cup \{v\}$ has label $\ell'(x) = 0$ and each vertex $y \in V(H') \setminus (S \cup \{v\})$ incident with an old trail edge has label $\ell(y) = 0$;

If the answers to tests (4) and (5) are both affirmative, the algorithm has detected a non-trivial closed trail T of H (as we prove in Theorem 7.9 below). We may assume that the algorithm also *finds* T, since that only requires some extra "bookkeeping" during the dynamic programming phase; we omitted the details for clarity of presentation. The algorithm then checks how many edges of H are dominated by T by computing $\beta(T)$. If $\beta(T) = |E(H)|$, then T is a dominating closed trail of H. Since every dominating closed trail is an optimum closed trail, the algorithm outputs T. Otherwise the algorithm stores T, unless it has already found a non-trivial closed trail T' with $\beta(T') \geq \beta(T)$ before, in which case T is discarded. If $k < |V(H')| - 1$, the algorithm repeats the above procedure for all sets $S \subseteq V(H')$ of size $k + 1$, all vertices $v \in V(H') \setminus S$ and all $\{0, 1\}$-labelings ℓ' of $S \cup \{v\}$. If $k = |V(H')| - 1$, then the algorithm terminates.

We now show that our first algorithm for finding an OCT is correct.

Theorem 7.9 (Correctness)**.** *When run on a connected graph H, the algorithm returns an optimum closed trail of H.*

Proof. As shown in the outline at the beginning of Section 7.4, the algorithm starts by checking if the input graph H is a tree. If H is a tree, then every closed trail of H consists of a single vertex. In particular, an optimum closed trail of H consists of a single vertex v of degree $\Delta(H)$. Hence the algorithm correctly outputs v in this case. If H is not a tree, then H contains a non-trivial closed trail; in particular, H contains an ONCT. We show below that the algorithm in fact finds such an ONCT T of H by executing the branching and dynamic programming procedures described in Section 7.4.1 and Section 7.4.2, respectively. Since an OCT of H might consist of a single vertex even if H is not a tree, the ONCT T is not necessarily an OCT of H. Hence the algorithm checks if a vertex v of maximum degree in H dominates more edges of H than T does. If so, then v is an OCT of H, and the algorithm correctly outputs v. Otherwise T is both an ONCT and an OCT of H, so the algorithm correctly outputs T.

It remains to show that, in case H is not a tree, the algorithm finds an ONCT T of H by executing the branching and dynamic programming procedures described in Section 7.4.1 and Section 7.4.2, respectively. Note that H has an optimum non-trivial closed trail by our assumption that H is not a tree.

We first show that if the algorithm computes $\beta(T)$ for a subgraph T of H, then T is a non-trivial closed trail of H. Only if the algorithm has found a stage-2 tuple $(H', W(H'), \ell)$ with some option (S, ℓ) for which the answers to tests (4) and (5) are both affirmative, it computes $\beta(T)$ for a subgraph T of H consisting of all old trail edges in $W(H')$ plus all new trail edges that have been added between the vertices of S. The dynamic programming, together with tests (3) and (4), ensures that T is connected. Tests (1), (2) and (5) together with the definition of a stage-2 tuple ensure that T is even. Hence, every subgraph T of H for which the value of $\beta(T)$ is computed is a non-trivial closed trail of H. Note that the algorithm does not compute $\beta(T)$ for *each* non-trivial closed trail T of H, but only for those that can be "built up" satisfying certain connectivity conditions throughout the dynamic programming phase.

It remains to show that the algorithm always finds an *optimum* non-trivial closed trail T of H. Due to Corollary 7.6 we may assume that T is 3-ordered. We show that our algorithm stores T, unless it has already stored another optimum non-trivial closed trail of H before it finds T. Let V' consist of all vertices that are not removed in the branching procedure, so $V' := V(H')$ for the graph H' in every stage-2 tuple. Let T' be the subgraph of T with $V(T') = V(T) \cap V'$. Then there exists a stage-2 tuple $(H', W(H'), \ell)$ such that $W(H')$ is exactly the set of edges of T that are incident with at least one vertex in $V(T) \setminus V'$, and such that $\ell(v) = 0$ if $v \in V' \setminus V(T')$, and $\ell(v) = 0$ (respectively $\ell(v) = 1$) if $v \in V(T')$ and v is incident with an even (respectively odd) number of edges in $W(H')$. Since our algorithm considers all possible stage-2 tuples, it will detect tuple $(H', W(H'), \ell)$. As T is 3-ordered, each component of T' is 3-ordered. This means that our dynamic programming procedure, based on the number of ways a vertex can be made adjacent to a set S with at most three edges, will find a labeling ℓ' such that (T_i, ℓ') is an option for each component T_i of T. As these components are connected to each other via old trail edges, at some moment (T', ℓ) will

be formed. Then tests (1)-(5) will all be successful and the algorithm will compute $\beta(T)$ for the subgraph T. Since T is an optimum non-trivial closed trail of H, there is no other non-trivial closed trail of H that dominates more edges of H than T does. Hence the algorithm will store T, unless it has encountered a non-trivial closed trail T' of H with $\beta(T') = \beta(T)$ before it found T, in which case the algorithm has stored T'. □

Below we give the overall time complexity of our first algorithm for solving the OCT problem.

Theorem 7.10 (Running time)**.** *The algorithm runs in $\mathcal{O}^*(1.6818^n)$ time on a connected graph with n edges.*

Proof. From the outline of the algorithm at the beginning of Section 7.4 it is clear that all steps not involving finding an ONCT can be performed in polynomial time. Hence it suffices to prove that the algorithm finds an ONCT of a connected graph H other than a tree in $\mathcal{O}^*(1.6818^n)$ time, where $n = |E(H)|$.

We first prove that the dynamic programming procedure presented at the beginning of Section 7.4.2 runs in $\mathcal{O}^*(3^p)$ time on any p-vertex graph. Let H' be a graph on p vertices. There are $\binom{p}{k}$ sets $S \subseteq V(H')$ of cardinality k, each of those sets has 2^k possible labelings ℓ', and there are $\binom{k}{0} + \binom{k}{1} + \binom{k}{2} + \binom{k}{3} = \mathcal{O}(k^3)$ ways to attach a new vertex v to a subset of cardinality k by using at most 3 edges. Each of the tests (1)-(5) can be done in polynomial time, and the same holds for computing $\beta(T)$ for a non-trivial closed trail T of H. Hence the time complexity of this procedure is

$$\mathcal{O}^*\Big(\sum_{k=1}^{p}\binom{p}{k}\cdot 2^k\cdot\mathcal{O}(k^3)\Big) = \mathcal{O}^*\Big(\sum_{k=0}^{p}\binom{p}{k}\cdot 1^{p-k}\cdot 2^k\Big) = \mathcal{O}^*\Big((1+2)^p\Big) = \mathcal{O}^*(3^p).$$

Let H be an instance of the OCT problem having n edges, and suppose H is not a tree. The algorithm repeatedly branches on vertices of degree at most $d^* = 4$. Let n_1 be the number of edges deleted during this branching phase. Then we obtain $\mathcal{O}^*(1.6818^{n_1})$ stage-2 tuples by Lemma 7.8. Let $(H', W(H'), \ell)$ be such a stage-2 tuple, where H' is a graph of minimum degree 5 having $n_2 := n - n_1$ edges and, say, p vertices. As shown above,

the dynamic programming procedure uses $\mathcal{O}^*(3^p)$ time. Since the minimum degree in H' is 5, we obtain $n_2 \geq 5p/2$, or equivalently $p \leq 2n_2/5$. Hence we can process each stage-2 tuple in time $\mathcal{O}^*(3^{\frac{2n_2}{5}}) = \mathcal{O}^*(1.5519^{n_2})$. This means that the overall time complexity of our algorithm on a graph H having $n = n_1 + n_2$ edges is

$$\mathcal{O}^*(1.6818^{n_1} \cdot 1.5519^{n_2}) = \mathcal{O}^*(1.6818^n).$$

The above time complexity is no longer guaranteed if we choose $d^* \neq 4$. □

Theorem 7.9 and Theorem 7.10 immediately imply the following.

Corollary 7.11. *The OCT problem for a connected graph H with n edges can be solved in $\mathcal{O}^*(1.6818^n)$ time, using exponential space.*

7.4.3 An $\mathcal{O}^*(1.8878^n)$ Time Algorithm

We describe our second algorithm for solving the OCT problem in the proof of the following theorem.

Theorem 7.12. *The OCT problem for a connected graph H with n edges can be solved in $\mathcal{O}^*(1.8878^n)$ time, using polynomial space.*

Proof. The second algorithm strongly resembles the first algorithm, described in Section 7.4.2. The only difference is the way in which the algorithm finds an ONCT of H in case H is not a tree. In order to prove correctness of our second algorithm for the OCT problem, it therefore suffices to prove correctness of the procedure of finding an ONCT of H described below.

Let H be a connected graph other than a tree. The algorithm executes the branching procedure described in Section 7.4.1, but this time we perform branching on vertices of degree at most $d^* = 12$. Suppose we delete n_1 edges during the branching process. By Lemma 7.8, this yields $\mathcal{O}^*(2^{11n_1/12}) = \mathcal{O}^*(1.8878^{n_1})$ stage-2 tuples $(H', W(H'), \ell)$, where each graph H' has p vertices of degree at least 13 and $n_2 = n - n_1$ edges. Note that $n_2 \geq 13p/2$, or equivalently $p \leq 2n_2/13$.

Since we assumed H not to be a tree, H has an ONCT T. By Theorem 7.6 we may assume that T is 2-degenerate. Let T' denote the (2-degenerate) subgraph of T that remains after the branching procedure. Note that T' is

a subgraph of the graph H' of some stage-2 tuple $(H', W(H'), \ell)$. It is well-known that any 2-degenerate graph on p vertices has at most $2p$ (or, more precisely, at most $2p-3$) edges. For every stage-2 tuple $(H', W(H'), \ell)$, we check for every possible subset $S \subseteq E(H')$ of edges up to cardinality $2p$ whether S together with the old trail edges in $W(H')$ forms a non-trivial closed trail T of H. If so, then we compute the number $\beta(T)$ of edges of H dominated by T, which can be done in polynomial time. If $\beta(T) = |E(H)|$, then T is a dominating closed trail of H. Since every dominating closed trail is an optimum closed trail, the algorithm outputs T. Otherwise the algorithm stores T, unless it has already found a non-trivial closed trail T' with $\beta(T') \geq \beta(T)$ before, in which case T is discarded. Since we check all subsets $S \subseteq E(H')$ for every stage-2 tuple $(H', W(H'), \ell)$, we are guaranteed to find an optimum non-trivial closed trail of H. This proves that our second algorithm for the OCT problem is correct.

From the outline of the algorithm at the beginning of Section 7.4 it is clear that all steps not involving finding an ONCT can be performed in polynomial time. Since the above procedure for finding an ONCT evidently only uses polynomial space, it remains to determine the time complexity of this procedure. Using Stirling's approximation $n_2! \approx n_2^{n_2} e^{-n_2} \sqrt{2\pi n_2}$ and the fact that $p \leq 2n_2/13$, the total number of checks per stage-2 tuple can be estimated as follows:

$$\sum_{k=1}^{2p} \binom{n_2}{k} \leq 2p \binom{n_2}{2p} \leq 2p \binom{n_2}{\frac{4n_2}{13}} = \mathcal{O}^*\Big(\Big(\frac{1}{\alpha^\alpha (1-\alpha)^{1-\alpha}}\Big)^{n_2}\Big) ,$$

where $\alpha = 4/13$, which constitutes $\mathcal{O}^*(1.8539^{n_2})$ checks. Since each of those checks can be performed in polynomial time and the number of stage-2 tuples we have to process is $\mathcal{O}^*(1.8878^{n_1})$, the overall time complexity of our second algorithm is

$$\mathcal{O}^*(1.8878^{n_1} \cdot 1.8539^{n_2}) = \mathcal{O}^*(1.8878^{n}) .$$

If we choose $d^* \neq 12$, then this time complexity is no longer guaranteed. □

7.5 Conclusion

We presented the first exact algorithms breaking the time complexity barrier of $\mathcal{O}^*(2^n)$ for the Longest Cycle problem on claw-free graphs. Our first algorithm uses $\mathcal{O}^*(1.6818^n)$ time and exponential space, and our second algorithm uses $\mathcal{O}^*(1.8878^n)$ time and polynomial space. A natural question is whether similar approaches can be used for other generalizations of the Hamiltonian Cycle problem.

Since a hamiltonian cycle is a connected 2-factor, the related NP-hard problem of determining a 2-factor with the smallest number of components in a claw-free graph G seems an obvious candidate. It was shown by Broersma and Paulusma [51] that this problem is equivalent to finding a smallest set of edge-disjoint stars with at least three edges and non-trivial closed trails that together dominate all edges of the preimage graph H of the closure of G. However, the approach in Section 7.4 for finding an OCT of H does not generalize in a straightforward way to finding such a smallest set of edge-disjoint stars with at least three edges and non-trivial closed trails. In fact, we do not believe a similar approach is possible because the counterpart of Section 7.4 would involve solving the following problem, which turns out to be NP-hard.

Decomposition in $\geq$ 3-Stars and Closed Trails (DEC)
Instance: a connected graph H.
Task: find a decomposition (partition) of $E(H)$ into stars with at least three edges and non-trivial closed trails.

It is not difficult to prove that the DEC problem is NP-hard by a reduction from the following decision problem, which is known to be NP-complete [200].

Decomposition in $\geq$ 3-Stars (DECOMP)
Instance: a connected graph H.
Question: can $E(H)$ be decomposed into stars with at least three edges?

Let G be an instance of the DECOMP problem. Replace each edge uv of G by the gadget illustrated in Figure 7.2, i.e., replace uv by a graph with vertex set $\{u, a, b, c, d, e, f, v\}$ and edge set $\{ua, ab, bc, cd, de, ef, fv, ae, bf\}$. Then, considering the edge cd, one readily checks that $E(G)$ has a decomposition into stars with at least three edges if and only if the newly constructed graph

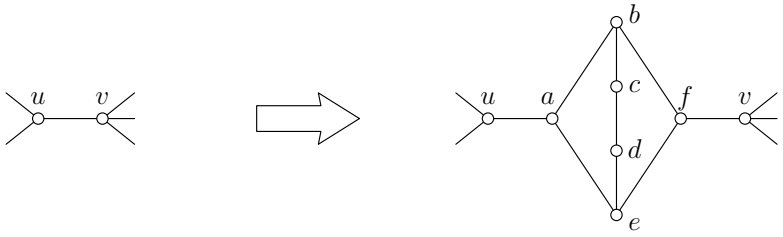

Figure 7.2: The gadget for replacing the edges of G.

has a decomposition of its edge set into stars with at least three edges and non-trivial closed trails. This shows that the decision version of the DEC problem is NP-complete, and hence that the DEC problem is NP-hard. Note that the construction shows that the DEC problem remains NP-hard even when restricted to 2-degenerate graphs.

Since a hamiltonian cycle is a connected 2-regular spanning subgraph, another direction would be to consider the problem of finding a connected spanning 3-regular subgraph of a claw-free graph. We have no idea how to generalize our approach in order to solve this problem. We heavily relied on the closure technique and the relationship between longest cycles in a claw-free graph G and optimum closed trails in the preimage graph H of the closure of G. We think it is highly unlikely that there is a natural counterpart of our approach for finding connected 3-regular spanning subgraphs.

Another interesting open problem is whether we can solve the Traveling Salesman problem for claw-free graphs in $\mathcal{O}^*(\alpha^n)$ time for some constant $\alpha < 2$. This also requires some new ideas, as our current approach involving the relationship between longest cycles in a claw-free graph G and optimum closed trails in the preimage graph H of the closure of G does not suffice.

Can we find an $\mathcal{O}^*(\alpha^n)$ time algorithm that solves the Hamiltonian Cycle problem for some constant $\alpha < 2$ for the class of bipartite graphs, or equivalently (cf. [215]) for the class of split graphs, or a superclass of split graphs such as the class of P_5-free graphs? As the Hamiltonian Cycle problem is already NP-complete for chordal bipartite graphs [215], this question is interesting for that class as well. One might also try to design fast exact algorithms for the Hamiltonian Cycle problem restricted to superclasses of claw-free graphs such as $K_{1,4}$-free graphs.

Bibliography

[1] J. Abello, M.R. Fellows, and J.C. Stillwell. On the complexity and combinatorics of covering finite complexes. *Australian Journal of Combinatorics*, 4:103–112, 1991.

[2] N. Alon, F.V. Fomin, G. Gutin, M. Krivelevich, and S. Saurabh. Parameterized algorithms for directed maximum leaf problems. In: *Proceedings of the 34th International Colloquium on Automata, Languages and Programming (ICALP 2007)*, volume 4596 of *Lecture Notes in Computer Science*, pages 352–262, Springer, 2007.

[3] M. Andersson, J. Gudmundsson, and C. Levcopoulos. Restricted mesh simplification using edge contraction. In: *Proceedings of the 12th Annual International Computing and Combinatorics Conference*, volume 4112 of *Lecture Notes in Computer Science*, pages 196–204, Springer, 2006.

[4] D. Angluin. Local and global properties in networks of processors. In: *Proceedings of the 12th Annual ACM Symposium on Theory of Computing (STOC 1980)*, pages 82–93, 1980.

[5] D. Angluin and A. Gardiner. Finite common coverings of pairs of regular graphs. *Journal of Combinatorial Theory, Series B*, 30:184–187, 1981.

[6] D.L. Applegate, R.E. Bixby, V. Chvátal, and W. Cook. *The Traveling Salesman Problem: A Computational Study.* Princeton University Press, Princeton, New Jersey, USA, 2006.

[7] D.L. Applegate, R.E. Bixby, V. Chvátal, W. Cook, D.G. Espinoza, M. Goycoolea, and K. Helsgaun. Certification of an optimal TSP tour through 85,900 cities. *Operations Research Letters*, 37(1):11–15, 2009.

[8] C. Arbib and R. Mosca. On (P_5, diamond)-free graphs. *Discrete Mathematics*, 250:1–22, 2002.

[9] S.R. Arikati and U.N. Peled. A linear algorithm for the group path problem on chordal graphs. *Discrete Applied Mathematics*, 44(1-3):185–190, 1993.

[10] S.R. Arikati and U.N. Peled. A polynomial algorithm for the parity path problem on perfectly orientable graphs. *Discrete Applied Mathematics*, 65(1):5–20, 1996.

[11] S.R. Arikati, C.P. Rangan, and G.K. Manacher. Efficient reduction for path problems on circular-arc graphs. *BIT Numerical Mathematics*, 31(2):182–193, 1991.

[12] E.M. Arkin, C.H. Papadimitriou, and M. Yannakakis. Modularity of cycles and paths in graphs. *Journal of the ACM*, 38(2):255–274, 1991.

[13] 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.

[14] G. Bacsó, D. Michalak, and Zs. Tuza. Dominating bipartite subgraphs in graphs. *Discussiones Mathematicae Graph Theory*, 25:85–94, 2005.

[15] G. Bacsó and Zs. Tuza. Dominating cliques in P_5-free graphs. *Periodica Mathematica Hungarica*, 21:303–308, 1990.

[16] G. Bacsó and Zs. Tuza. Dominating subgraphs of small diameter. *Journal of Combinatorics, Information and System Sciences*, 22(1):51–62, 1997.

[17] S. Baumer and R. Schuler. Improving a probabilistic 3-SAT algorithm by dynamic search and independent clause pairs. In: *Theory and Applications of Satisfiability Testing*, volume 2919 of *Lecture Notes in Computer Science*, pages 150–161, Springer, 2003.

[18] E.T. Bax. Inclusion and exclusion algorithm for the Hamiltonian path problem. *Information Processing Letters*, 47:203–207, 1993.

[19] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database systems. *Journal of the ACM*, 30:479–513, 1983.

[20] R. Beigel and D. Eppstein. 3-coloring in time $\mathcal{O}(1.3446^n)$: A no-MIS algorithm. In: *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS 1995)*, pages 444–452, IEEE Computer Society, 1995.

[21] R. Beigel. Finding maximum independent sets in sparse and general graphs. In: *Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms (SODA 1999)*, pages 856–857, 1999.

[22] R. Beigel. Improved algorithms 3-coloring, 3-edge-coloring, and constraint satisfaction. In: *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms (SODA 2001)*, pages 329–337, 2001.

[23] R. Beigel and D. Eppstein. 3-coloring in time $\mathcal{O}(1.3289^n)$. *Journal of Algorithms*, 54:168–204, 2005.

[24] L.W. Beineke. Characterizations of derived graphs. *Journal of Combinatorial Theory, Series B*, 9:129–135, 1970.

[25] C. Berge. Färbung von Graphen, deren sämtliche bzw. deren ungerade Kreise starr sind. *Wissenschaftliche Zeitschrift der Martin-Luther-Universität Halle-Wittenberg, Mathematisch-Naturwissenschaftliche Reihe*, 10:114, 1961.

[26] M.W. Bern, E.L. Lawler, and A.L. Wong. Linear-time computation of optimal subgraphs of decomposable graphs. *Journal of Algorithms*, 8(2):216–235, 1987.

[27] A. Berry, E. Dahlhaus, P. Heggernes, and G. Simonet. Sequential and parallel triangulating algorithms for Elimination Game and new insights on Minimum Degree. *Theoretical Computer Science*, 409(3):601–616, 2008.

[28] A.A. Bertossi. The edge hamiltonian path problem is NP-complete. *Information Processing Letters*, 13:157–159, 1981.

[29] A.A. Bertossi. Dominating sets for split and bipartite graphs. *Information Processing Letters*, 19(1):37–40, 1984.

[30] A.A. Bertossi and M.A. Bonuccelli. Hamiltonian circuits in interval graph generalizations. *Information Processing Letters*, 23:195–200, 1986.

[31] N.L. Biggs. Constructing 5-arc transitive cubic graphs. *Journal of the London Mathematical Society (2)*, 26:193–200, 1982.

[32] D. Bienstock. On the complexity of testing for odd holes and induced odd paths. *Discrete Mathematics*, 90(1):85–92, 1991.

[33] A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. The travelling salesman problem in bounded degree graphs. In: *Proceedings of the 35th International Colloquium on Automata, Languages and Programming (ICALP 2008)*, volume 5125 of *Lecture Notes in Computer Science*, pages 198–209, Springer, 2008.

[34] A. Björklund, T. Husfeldt, and M. Koivisto. Set partitioning via inclusion-exclusion. *SIAM Journal on Computing*, 39(2):546–563, 2009.

[35] J.R.S. Blair and B.W. Peyton. An introduction to chordal graphs and clique trees. In: *Sparse Matrix Computations: Graph Theory Issues and Algorithms*, volume 56 of *IMA Volumes in Mathematics and its Applications*, pages 1–30, Springer, 1993.

[36] D. Blum. *Circularity of graphs.* PhD thesis, Virginia Polytechnic Institute and State University, 1982.

[37] H.L. Bodlaender. Dynamic programming on graphs with bounded treewidth. In: *Proceedings of the 15th International Colloquium on Automata, Languages and Programming (ICALP 1988)*, volume 317 of *Lecture Notes in Computer Science*, pages 105–118, Springer, 1988.

[38] H.L. Bodlaender. The classification of coverings of processor networks. *Journal of Parallel Distributed Computing*, 6:166–182, 1989.

[39] H.L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In: *Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC 1993)*, pages 226–234, 1993.

[40] H.L. Bodlaender. Discovering Treewidth. In: *Proceedings of the 31st Annual Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2005)*, volume 3381 of *Lecture Notes in Computer Science*, pages 1–16, Springer, 2005.

[41] R. Boliac and V.V. Lozin. An augmenting graph approach to the stable set problem in P_5-free graphs. *Discrete Applied Mathematics*, 131:567–575, 2003.

[42] K.S. Booth and J.H. Johnson. Dominating sets in chordal graphs. *SIAM Journal on Computing*, 11(1):191–199, 1982.

[43] N. Bourgeois, B. Escoffier and V.Th. Paschos. Approximation of min coloring by moderately exponential algorithms. *Information Processing Letters*, 109(16):950–954, 2009.

[44] A. Brandstädt, J. Engelfriet, H.O. Le, and V.V. Lozin. Clique-width for 4-vertex forbidden subgraphs. *Theory of Computing Systems*, 39(4): 561–590, 2006.

[45] A. Brandstädt and C.T. Hoàng. On clique separators, nearly chordal graphs, and the maximum weight stable set problem. *Theoretical Computer Science*, 389:295–306, 2007.

[46] A. Brandstädt, T. Klembt, and S. Mahfud. P_6- and triangle-free graphs revisited: structure and bounded clique-width. *Discrete Mathematics and Theoretical Computer Science*, 8:173–188, 2006.

[47] A. Brandstädt, V.B. Le, and J. Spinrad. *Graph Classes: A Survey.* SIAM Monographs on Discrete Mathematics and Applications 3, SIAM, Philadelphia, 1999.

[48] A. Brandstädt and R. Mosca. On the structure and stability number of P_5- and co-chair-free graphs. *Discrete Applied Mathematics*, 132:47–65, 2004.

[49] H.J. Broermsa, F.V. Fomin, P.A. Golovach, and D. Paulusma. Three complexity results on coloring P_k-free graphs. In: *Proceedings of the 20th International Workshop on Combinatorial Algorithms (IWOCA 2009)*, volume 5875 of *Lecture Notes in Computer Science*, pages 95–104, 2009.

[50] H.J. Broersma, F.V. Fomin, P. van 't Hof, and D. Paulusma. Fast exact algorithms for hamiltonicity in claw-free graphs. In: *Proceedings of the 35th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2009)*, volume 5911 of *Lecture Notes in Computer Science*, pages 44–53, Springer, 2009.

[51] H.J. Broersma and D. Paulusma. Computing sharp 2-factors in claw-free graphs. In: *Proceedings of the 33th International Symposium on Mathematical Foundations of Computer Science (MFCS 2008)*, volume 5162 of *Lecture Notes in Computer Science*, pages 193–204, Springer, 2008.

[52] A.E. Brouwer and H.J. Veldman. Contractibility and NP-completeness. *Journal of Graph Theory*, 11:71–79, 1987.

[53] D. Bruce, C.T. Hoàng, and J. Sawada. A certifying algorithm for 3-colorability of P_5-free graphs. In: *Proceedings of the 20th International Symposium on Algorithms and Computation (ISAAC 2009)*, volume 5878 of *Lecture Notes in Computer Science*, pages 594–604, Springer, 2009.

[54] T. Brueggemann and W. Kern. An improved deterministic local search algorithm for 3-SAT. *Theoretical Computer Science*, 329:303–313, 2004.

[55] P. Buneman. The recovery of trees from measures of dissimilarity. In: *Mathematics in the Archaeological and Historical Sciences*, pages 387–395, Edinburgh University Press, Edinburgh, 1972.

[56] P. Buneman. A characterization of rigid circuit graphs. *Discrete Mathematics*, 9:205–212, 1974.

[57] J.M. Byskov. Enumerating maximal independent sets with applications to graph colouring. *Operations Research Letters*, 32:547–556, 2004.

[58] J. Chalopin, Y. Métivier, and W. Zielonka. Local computations in graphs: the case of cellular edge local computations. *Fundamenta Informaticae*, 74:85–114, 2006.

[59] L.S. Chandran and F. Grandoni. Refined memorization for vertex cover. *Information Processing Letters*, 93:125–131, 2005.

[60] S. Cheng, T. Dey, and S. Poon. Hierarchy of surface models and irreducible triangulations. *Computational Geometry Theory and Applications*, 27:135–150, 2004.

[61] J. Chen, I.A. Kanj, and W. Jia. Vertex cover: further observations and further improvements. *Journal of Algorithms*, 41:280–301, 2001.

[62] M. Chudnovsky, G. Cornuéjols, X. Liu, P.D. Seymour, and K. Vušković. Recognizing Berge Graphs. *Combinatorica*, 25(2):143–186, 2005.

[63] M. Chudnovsky, K. Kawarabayashi, and P.D. Seymour. Detecting even holes. *Journal of Graph Theory*, 48(2):85–111, 2005.

[64] M. Chudnovsky, N. Robertson, P.D. Seymour, and R. Thomas. The Strong Perfect Graph Theorem. *Annals of Mathematics*, 164:51–229, 2006.

[65] M. Chudnovsky and P.D. Seymour. The structure of claw-free graphs. *Surveys in Combinatorics 2005*, volume 327 of *London Mathematical Society Lecture Note Series*, pages 153–171, 2005.

[66] M. Chudnovsky and P.D. Seymour. Claw-free graphs I. Orientable prismatic graphs. *Journal of Combinatorial Theory, Series B*, 97:867–901, 2007.

[67] M. Chudnovsky and P.D. Seymour. Claw-free graphs II. Non-orientable prismatic graphs. *Journal of Combinatorial Theory, Series B*, 98:249–290, 2008.

[68] M. Chudnovsky and P.D. Seymour. Claw-free graphs III. Circular interval graphs. *Journal of Combinatorial Theory, Series B*, 98:812–834, 2008.

[69] M. Chudnovsky and P.D. Seymour. Claw-free graphs IV. Decomposition theorem. *Journal of Combinatorial Theory, Series B*, 98:839–938, 2008.

[70] M. Chudnovsky and P.D. Seymour. Claw-free graphs V. Global structure. *Journal of Combinatorial Theory, Series B*, 98:1373–1410, 2007.

[71] M. Chudnovsky and P.D. Seymour. Claw-free graphs VI. Coloring claw-free graphs. Submitted for publication, manuscript available at `http://www.columbia.edu/~mc2775/publications.html`.

[72] M. Chudnovsky and P.D. Seymour. Claw-free graphs VII. Quasi-line graphs. Submitted for publication, manuscript available at `http://www.columbia.edu/~mc2775/publications.html`.

[73] M. Chudnovsky and P.D. Seymour. The three-in-a-tree problem. *Combinatorica*, to appear, manuscript available at `http://www.columbia.edu/~mc2775/publications.html`.

[74] M. Chudnovsky and P.D. Seymour. Three-colourable perfect graphs without even pairs. Submitted for publication, manuscript available at `http://www.columbia.edu/~mc2775/publications.html`.

[75] F.R.K. Chung and D. Mumford. Chordal completions of planar graphs. *Journal of Combinatorial Theory, Series B*, 62(1):96–106, 1994.

[76] V. Chvátal and N. Sbihi. Recognizing claw-free perfect graphs. *Journal of Combinatorial Theory, Series B*, 44:154–176, 1988.

[77] J. Cong and S.K. Lim. Edge separability-based circuit clustering with application to multilevel circuit partitioning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23:346–357, 2004.

[78] S.A. Cook. The complexity of theorem proving procedures. In: *Proceedings of the 3rd Annual ACM Symposium of Theory of Computing (STOC 1971)*, pages 151–158, 1971.

[79] W. Cook, D.G. Espinoza, and M. Goycoolea. Computing with domino-parity inequalities for the travelling saleman problem (TSP). *INFORMS Journal on Computing*, 19(3), 356–365, 2007.

[80] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. Second edition, MIT Press/McGraw-Hill, 2001.

[81] D.G. Corneil and J. Fonlupt. Stable set bonding in perfect graphs and parity graphs. *Journal of Combinatorial Theory, Series B*, 59:1–14, 1993.

[82] D.G. Corneil, H. Lerchs, and L. Stewart Burlingham. Complement reducible graphs. *Discrete Applied Mathematics*, 3:163–174, 1981.

[83] D.G. Corneil, Y. Perl, and L.K. Stewart. A linear recognition algorithm for cographs. *SIAM Journal on Computing*, 14(4):926–934, 1985.

[84] B. Courcelle. The monadic second-order logic of graphs I. Recognizable sets of finite graphs. *Information and Computation*, 85:12–75, 1990.

[85] M.B. Cozzens and L.L. Kelleher. Dominating cliques in graphs. *Discrete Mathematics*, 86:101–116, 1990.

[86] M. Cygan, L. Kowalik, and M. Wykurz. Exponential-time approximation of weighted set cover. *Information Processing Letters*, 109(16):957–961, 2009.

[87] V. Dahllöf and P. Jonsson. An algorithm for counting maximum weighted independent sets and its applications. In: *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2002)*, pages 292–298, 2002.

[88] E. Dantsin, A. Goerdt, E.A. Hirsch, R. Kannan, J. Kleinberg, C. Papadimitriou, P. Raghavan, and U. Schöning. A deterministic $2 - \frac{2}{k+1}$ algorithm for k-SAT based on local search. *Theoretical Computer Science*, 289(1):69–83, 2002.

[89] E.D. Demaine and M. Hajiaghayi. The bidimensionality theory and its algorithmic applications. *Computer Journal*, 51(3):292–302, 2008.

[90] N. Derhy and C. Picouleau. Finding induced trees. *Discrete Applied Mathematics*, 157:3552–3557, 2009.

[91] R. Diestel. *Graph Theory.* Third edition, Springer-Verlag Heidelberg, 2005.

[92] G.A. Dirac. On rigid circuit graphs. *Abhandlungen Mathematischen Seminar Universität Hamburg*, 25:71–76, 1961.

[93] F. Dorn, F.V. Fomin, and D.M. Thilikos. Catalan structures and dynamic programming on H-minor-free graphs. In: *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2008)*, pages 631–640, 2008.

[94] F. Dorn, E. Penninkx, H. Bodlaender, and F.V. Fomin. Efficient exact algorithms on planar graphs: Exploiting sphere cut branch decompositions. In: *Proceedings of the 13th Annual European Symposium on Algorithms (ESA 2005)*, volume 3669 of *Lecture Notes in Computer Science*, pages 95–106, Springer, 2005.

[95] R.G. Downey and M.R. Fellows. *Parameterized Complexity.* Monographs in Computer Science, Springer, 1999.

[96] J. Edmonds. Paths, trees and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[97] D. Eppstein. The traveling salesman problem for cubic graphs. In: *Proceedings of the 8th Workshop on Algorithms and Data Structures (WADS 2003)*, volume 2748 of *Lecture Notes in Computer Science*, pages 307–318, Springer, 2003.

[98] D. Eppstein. The traveling salesman problem for cubic graphs. *Journal of Graph Algorithms and Applications*, 11:61–81, 2007.

[99] M.G. Everett and S. Borgatti. Role colouring a graph. *Mathematical Social Sciences*, 21:183–188, 1991.

[100] H. Everett, C.M.H. de Figueiredo, C. Linhares Sales, F. Maffray, O. Porto, and B.A. Reed. Path parity and perfection. *Discrete Mathematics*, 165-166:233–252, 1997.

[101] H. Everett, C.M.H. de Figueiredo, C. Linhares Sales, F. Maffray, O. Porto, B.A. Reed. In: *Perfect Graphs*, pages 67–92, Wiley, 2001.

[102] R. Faudree, E. Flandrin, and Z. Ryjáček. Claw-free graphs — A survey. *Discrete Mathematics*, 164:87–147, 1997.

[103] M.R. Fellows, J. Kratochvíl, M. Middendorf, and F. Pfeiffer. The complexity of induced minors and related problems. *Algorithmica*, 13:266–282, 1995.

[104] J. Fiala, M. Kamiński, B. Lidický, and D. Paulusma. The k-in-a-path problem for claw-free graphs. In: *Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science (STACS 2010)*, pages 371–382, 2010, available at `http://dx.doi.org/10.4230/LIPIcs.STACS.2010.2469`.

[105] J. Fiala and J. Kratochvíl. Complexity of partial covers of graphs. In: *Proceedings of the 12th International Symposium on Algorithms and Computation (ISAAC 2001)*, volume 2223 of *Lecture Notes in Computer Science*, pages 537–549, Springer, 2001.

[106] J. Fiala and J. Kratochvíl. Partial covers of graphs. *Discussiones Mathematicae Graph Theory*, 22:89–99, 2002.

[107] J. Fiala, J. Kratochvíl, and T. Kloks. Fixed-parameter complexity of λ-labelings. *Discrete Applied Mathematics*, 113:59–72, 2001.

[108] J. Fiala and D. Paulusma. A complete complexity classification of the role assignment problem. *Theoretical Computer Science*, 349:67–81, 2005.

[109] J. Fiala and D. Paulusma. Comparing universal covers in polynomial time. *Theory of Computing Systems*, 46:620–635, 2010.

[110] C.M.H. de Figueiredo, J.G. Gimbel, C.P. Mello, and J.L. Szwarcfiter. Even and odd pairs in comparability and in P_4-comparability graphs. *Discrete Applied Mathematics*, 91(1-3):293–297, 1999.

[111] S. Földes and P.L. Hammer. Split graphs. In: *Proceedings of the 8th South-Eastern Conference on Combinatorics, Graph Theory and Computing*, volume 19 of *Congressus Numerantium*, pages 311–315, Utilitas Mathematica, 1977.

[112] F.V. Fomin, P.A. Golovach, and D.M. Thilikos. Contraction bidimensionality: the accurate picture. In: *Proceedings of the 17th Annual Eu-*

ropean Symposium on Algorithms (ESA 2009), volume 5757 of *Lecture Notes in Computer Science*, pages 706–717, Springer, 2009.

[113] F.V. Fomin, F. Grandoni, and D. Kratsch. A measure & conquer approach for the analysis of exact algorithms. *Journal of the ACM*, 56(5):1–32, 2009.

[114] F.V. Fomin, F. Grandoni, and D. Kratsch. Some new techniques in design and analysis of exact (exponential) algorithms. *Bulletin of the EATCS*, 87:47–77, 2005.

[115] F.V. Fomin, F. Grandoni, and D. Kratsch. Measure and conquer: a simple $O(2^{0.288n})$ Independent Set algorithm. In: *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2006)*, 18–25, 2006.

[116] F.V. Fomin, F. Grandoni, and D. Kratsch. Solving connected dominating set faster than 2^n. *Algorithmica*, 52(2):153–166, 2008.

[117] F.V. Fomin and D. Kratsch. *Exact Algorithms*. In preparation (personal communication).

[118] J. Fonlupt and J.P. Uhry. Transformations which preserve perfectness and H-perfectness of graphs. *Annals of Discrete Mathematics*, 16:83–85, 1982.

[119] J.-L. Fouquet, V. Giakoumakis, F. Maire, and H. Thuillier. On graphs without P_5 and $\overline{P_5}$. *Discrete Mathematics*, 146:33–44, 1995.

[120] J.-L. Fouquet, V. Giakoumakis, and J.-M. Vanherpe. Bipartite graphs totally decomposable by canonical decomposition. *International Journal of Foundations of Computer Science*, 10(4):513–534, 1999.

[121] D.R. Fulkerson and O.A. Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 15:835–855, 1965.

[122] M. Fürer, S. Gaspers, and S.P. Kasiviswanathan. An exponential time 2-approximation algorithm for Bandwidth. In: *Proceedings of the 4th International Workshop on Parameterized and Exact Computation*

(IWPEC 2009), volume 5917 of *Lecture Notes in Computer Science*, pages 173–184, Springer, 2009.

[123] P. Galinier, M. Habib, and C. Paul. Chordal graphs and their clique trees. In: *Proceedings of the 21st International Workshop on Graph-Theoretic Concepts in Computer Science (WG 1995)*, volume 1017 of *Lecture Notes in Computer Science*, pages 358–371, Springer, 1995.

[124] P. Galinier and A. Hertz. A survey of local search methods for graph coloring. *Computers & Operations Research*, 33:2547–2562, 2006.

[125] M.R. Garey, R.L. Graham, and D.S. Johnson. Some NP-complete geometric problems. In: *Proceedings of the 8th Annual ACM Symposium on Theory of Computing (STOC 1976)*, pages 10–22, 1976.

[126] M.R. Garey, R.L. Graham, D.S. Johnson, and D.E. Knuth. Complexity results for bandwidth minimization. *SIAM Journal on Applied Mathematics*, 34:477–495, 1978.

[127] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Co., New York, 1979.

[128] S. Gaspers and M. Liedloff. A branch-and-reduce algorithm for finding a minimum independent dominating set in graphs. In: *Proceedings of the 32nd International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2006)*, volume 4271 of *Lecture Notes in Computer Science*, pages 78–89, Springer, 2006.

[129] F. Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques and maximum independent set of a chordal graph. *SIAM Journal on Computing*, 1:180–187, 1972.

[130] F. Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory, Series B*, 16:47–56, 1974.

[131] F. Gavril. Algorithms on clique separable graphs. *Discrete Mathematics*, 19:159–165, 1977.

[132] H. Gebauer. On the number of hamilton cycles in bounded degree graphs. In: *Proceedings of the 4th Workshop on Analytic and Combinatorics (ANALCO 2008)*, pages 241–248, SIAM, 2008.

[133] A. George and J.W.H. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall, New Jersey, 1981.

[134] M.U. Gerber, A. Hertz, and D. Schindl. P_5-free augmenting graphs and the maximum stable set problem. *Discrete Applied Mathematics*, 132:109–119, 2004.

[135] M.U. Gerber and V.V. Lozin. On the stable set problem in special P_5-free graphs. *Discrete Applied Mathematics*, 125:215–224, 2003.

[136] V. Giakoumakis and J.M. Vanherpe. Linear time recognition and optimizations for weak-bisplit graphs, bi-cographs and bipartite P_6-free graphs. *International Journal of Foundations of Computer Science*, 14:107–136, 2003.

[137] M.C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Second edition, Annals of Discrete Mathematics 57, Elsevier, North Holland, 2004.

[138] M.C. Golumbic and A.N. Trenk. *Tolerance graphs*. Cambridge University Press, Cambridge, 2003.

[139] S. Goodman and S. Hedetniemi. Sufficient conditions for a graph to be hamiltonian. *Journal of Combinatorial Theory, Series B*, 16:175–180, 1974.

[140] R.J. Gould and M.S. Jacobson. Forbidden subgraphs and hamiltonian properties in graphs. *Discrete Mathematics*, 42(2-3):189–196, 1982.

[141] F. Grandoni. A note on the complexity of mimimum dominating set. *Journal of Discrete Algorithms*, 4:209–214, 2006.

[142] C. Gray, F. Kammer, M. Löffler, and R.I. Silveira. Removing local extrema from imprecise terrains. In: *Abstracts of the 26th European Workshop on Computational Geometry (EuroCG 2010)*, full version available at `http://arxiv1.library.cornell.edu/abs/1002.2580`.

[143] M. Habib and C. Paul. A simple linear time algorithm for cograph recognition. *Discrete Applied Mathematics*, 145:183-197, 2005.

[144] L. Hagen and A.B. Kahng. New spectral methods for ratio cut partitioning and clustering. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(9):1074–1085, 1992.

[145] F. Harary. *Graph Theory*. Addison-Wesley, Reading MA, 1969.

[146] F. Harary and C. St. J.A. Nash-Williams. On eulerian and hamiltonian graphs and line graphs. *Canadian Mathematical Bulletin*, 8:701–709, 1965.

[147] F. Harary and R.Z. Norman. Some properties of line digraphs. *Rendiconti del Circulo Mathematico di Palermo*, 9:161–169, 1960.

[148] D. Harel and Y. Koren. On clustering using random walks. In: *Proceedings of the 21st Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2001)*, volume 2245 of *Lecture Notes in Computer Science*, pages 18–41, Springer, 2001.

[149] J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, 182(1):105–142, 1999.

[150] P. Heggernes, D. Lokshtanov, R. Mihai, and C. Papadopoulos. Cutwidth of split graphs, threshold graphs, and proper interval graphs. In: *Proceedings of the 34th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2008)*, volume 5344 of *Lecture Notes in Computer Science*, pages 218–229, Springer, 2008.

[151] M. Held and R.M. Karp. A dynamic programming approach to sequencing problems. *Journal of SIAM*, 10:196–210, 1962.

[152] C.T. Hoàng, M. Kamiński, V.V. Lozin, J. Sawada, and X. Shu. Deciding k-colorability of P_5-free graphs in polynomial time. *Algorithmica*, 57:74–81, 2010.

[153] C.T. Hoàng and V.B. Le. Recognizing perfect 2-split graphs. *SIAM Journal on Discrete Mathematics*, 13(1):48–55, 2000.

[154] C. Hoede and H.J. Veldman. Contraction theorems in Hamiltonian graph theory, *Discrete Mathematics*, 34:61–67, 1981.

[155] P. van 't Hof, M. Kamiński, and D. Paulusma. Finding induced paths of given parity in claw-free graphs. In: *Proceedings of the 35th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2009)*, volume 5911 of *Lecture Notes in Computer Science*, pages 341–352, Springer, 2009.

[156] P. van 't Hof, M. Kamiński, D. Paulusma, S. Szeider, and D.M. Thilikos. On contracting graphs to fixed pattern graphs. In: *Proceedings of the 36th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2010)*, volume 5901 of *Lecture Notes in Computer Science*, pages 503–514, Springer, 2010.

[157] P. van 't Hof and D. Paulusma. A new characterization of P_6-free graphs. In: *Proceedings of the 14th Annual International Computing and Combinatorics Conference (COCOON 2008)*, volume 5092 of *Lecture Notes in Computer Science*, pages 415–424, Springer, 2008.

[158] P. van 't Hof and D. Paulusma. A new characterization of P_6-free graphs. *Discrete Applied Mathematics*, 158(7):731–740, 2010.

[159] P. van 't Hof, D. Paulusma, and J.M.M. van Rooij. Computing role assignments of chordal graphs. In: *Proceedings of the 17th International Symposium on Fundamentals of Computation Theory (FCT 2009)*, volume 5699 of *Lecture Notes in Computer Science*, pages 193–204, Springer, 2009.

[160] P. van 't Hof, D. Paulusma, and G.J. Woeginger. Partitioning graphs into connected parts. In: *Proceedings of the 17th International Computer Science Symposium in Russia (CSR 2009)*, volume 5675 of *Lecture Notes in Computer Science*, pages 143–154, Springer, 2009.

[161] P. van 't Hof, D. Paulusma, and G.J. Woeginger. Partitioning graphs into connected parts. *Theoretical Computer Science*, 410:4834–4843, 2009.

[162] T. Hofmeister, U. Schöning, R. Schuler, and O. Watanabe. A probabilistic 3-SAT algorithm further improved. In: *Proceedings of the 19th Symposium on Theoretical Aspects of Computer Science (STACS 2002)*, volume 2285 of *Lecture Notes in Computer Science*, pages 193–202, Springer, 2002.

[163] I. Holyer. The NP-completeness of edge-coloring. *SIAM Journal on Computing*, 10:718–720, 1981.

[164] E. Horowitz and S. Sahni. Computing partitions with applications to the knapsack problem. *Journal of the ACM*, 21:277–292, 1974.

[165] W.-L. Hsu. How to color claw-free perfect graphs. *Annals of Discrete Mathematics*, 11:189–197, 1981.

[166] W.-L. Hsu. Recognizing planar perfect graphs. *Journal of the ACM*, 34(2):255–288, 1987.

[167] W.-L. Hsu and G.L. Nemhauser. Algorithms for minimum covering by cliques and maximum clique in claw-free perfect graphs. *Discrete Mathematics*, 37:181–191, 1981.

[168] R.Z. Hwang, R.C. Chang, and R.C.T. Lee. The searching over separators strategy to solve some NP-hard problems in subexponential time. *Algorithmica*, 9:398–423, 1993.

[169] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? In: *Proceedings of the 39th Annual Symposium on Foundations of Computer Science (FOCS 1998)*, pages 653–663, IEEE Computer Society, 1998.

[170] K. Iwama and T. Nakashima. An improved exact algorithm for cubic graph TSP. In: *Proceedings of the 13th Annual International Computing and Combinatorics Conference (COCOON 2007)*, volume 4598 of *Lecture Notes in Computer Science*, pages 108–117, Springer, 2007.

[171] K. Iwama and S. Tamaki. Improved upper bounds for 3-SAT. In: *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2004)*, pages 328–329, 2004.

[172] T. Jian. An $O(2^{0.304n})$ algorithm for solving Maximum Independent Set problem. *IEEE Transactions on Computers*, 35:847–851, 1986.

[173] F. Kammer and T. Tholey. The complexity of minimum convex coloring. In: *Proceedings of the 19th International Symposium on Algorithms and Computation (ISAAC 2008)*, volume 5369 of *Lecture Notes in Computer Science*, pages 16–27, Springer, 2008.

[174] I.A. Kanj and D. Kratsch. Convex recolorings revisited: complexity and exact algorithms. In: *Proceedings of the 15th International Computing and Combinatorics Conference (COCOON 2009)*, volume 5609 of *Lecture Notes in Computer Science*, pages 388–397, 2009.

[175] H. Kaplan, R. Shamir, and R.E. Tarjan. Tractability of parameterized completion problems on chordal, strongly chordal, and proper interval graphs. *SIAM Journal on Computing*, 28(5):1906–1922, 1999.

[176] R.M. Karp. Reducibility among combinatorial problems. In: *Complexity of Computer Computations*, pages 85–103, Plenum Press, New York, 1972.

[177] R.M. Karp. On the complexity of combinatorial problems. *Networks*, 5:45–68, 1975.

[178] R.M. Karp. Dynamic programming meets the principle of inclusion and exclusion. *Operations Research Letters*, 1:49–51, 1982.

[179] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20:359–392, 1999.

[180] A. King and B. Reed. Bounding χ in terms of ω and δ for quasi-line graphs. *Journal of Graph Theory*, 59:215–228, 2008.

[181] D.J. Kleitman and R.V. Vohra. Computing the bandwidth of interval graphs. *SIAM Journal on Discrete Mathematics*, 3(3):373–375, 1990.

[182] S. Kohn, A. Gottlieb, and M. Kohn. A generating function approach to the traveling salesman problem. In: *Proceedings of the ACM annual conference (ACM 1977)*, pages 294–300, ACM Press, 1977.

[183] D. Král', J. Kratochvíl, Zs. Tuza and G.J. Woeginger. Complexity of coloring graphs without forbidden induced subgraphs. In: *Proceedings of the 28th Workshop on Graph-Theoretic Concepts in Computer Science (WG 2001)*, volume 2204 of *Lecture Notes in Computer Science*, pages 254–262, 2001.

[184] J. Kratochvíl, A. Proskurowski, J.A. Telle. Covering regular graphs. *Journal of Combinatorial Theory, Series B*, 71:1–16, 1997.

[185] O. Kullmann. Worst-case analysis, 3-SAT decisions, and lower bounds: Approaches for improved SAT algorithms. In: *Satisfiability Problem: Theory and Applications*, volume 35 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 261–313, 1997.

[186] O. Kullmann. New methods for 3-SAT decision and worst case analysis. *Theoretical Computer Science*, 223:1–72, 1999.

[187] A.S. LaPaugh and C.H. Papadimitriou. The even-path problem for graphs and digraphs. *Networks*, 14:507–513, 1984.

[188] M. Las Vergnas. A note on matchings in graphs. *Cahiers du Centre d'Études de Recherche Opérationnelle*, 17(2-3-4):257–260, 1975.

[189] E.L. Lawler. A note on the complexity of the chromatic number problem. *Information Processing Letters*, 5:66–67, 1976.

[190] V.B. Le, B. Randerath, and I. Schiermeyer. On the complexity of 4-coloring graphs without long induced paths. *Theoretical Computer Science*, 389(1-2):330–335, 2007.

[191] H. Lerchs. On cliques and kernels. *Technical report*, Department of Computer Science, University of Toronto, 1971.

[192] B. Lévêque, D.Y. Lin, F. Maffray, and N. Trotignon. Detecting induced subgraphs. *Discrete Applied Mathematics*, 157(17):3540–3551, 2009.

[193] A. Levin, D. Paulusma, and G.J. Woeginger. The computational complexity of graph contractions I: polynomially solvable and NP-complete cases. *Networks*, 51:178–189, 2008.

[194] A. Levin, D. Paulusma, and G.J. Woeginger. The computational complexity of graph contractions II: two tough polynomially solvable cases. *Networks*, 52:32–56, 2008.

[195] M. Li, D.G. Corneil, and E. Mendelsohn. Pancyclicity and NP-completeness in planar graphs. *Discrete Applied Mathematics*, 98:219–225, 2000.

[196] X. Li and W. Zang. A combinatorial algorithm for minimum weighted colorings of claw-free perfect graphs. *Journal of Combinatorial Optimization*, 9(4):331–347, 2005.

[197] C. Linhares Sales and F. Maffray. Even pairs in claw-free perfect graphs. *Journal of Combinatorial Theory, Series B*, 74:169–191, 1998.

[198] J. Liu and H. Zhou. Dominating subgraphs in graphs with some forbidden structures. *Discrete Mathematics*, 135:163–168, 1994.

[199] J. Liu, Y. Peng, and C. Zhao. Characterization of P_6-free graphs. *Discrete Applied Mathematics*, 155:1038–1043, 2007.

[200] Z. Lonc. On the complexity of some edge-partition problems for graphs. *Discrete Applied Mathematics*, 70:177–183, 1996.

[201] L. Lovasz. Coverings and colorings of hypergraphs. In: *Proceedings of the 4th Southeastern Conference on Combinatorics, Graph Theory, and Computing*, pages 3–12, Utilitas Mathematica, 1973.

[202] V.V. Lozin. Stability in P_5- and banner-free graphs. *European Journal of Operational Research*, 125:292–297, 2000.

[203] V.V. Lozin and R. Mosca. Maximum independent sets in subclasses of P_5-free graphs. *Information Processing Letters*, 109:319–324, 2009.

[204] F. Maffray. Stable sets in k-colorable P_5-free graphs. *Information Processing Letters*, 109(23-24):1235–1237, 2009.

[205] F. Maffray and B.A. Reed. A description of claw-free perfect graphs. *Journal of Combinatorial Theory, Series B*, 75:134–156, 1999.

[206] N.V.R. Mahadev and U.N. Peled. *Threshold graphs and related topics.* North-Holland, Amsterdam, 1995.

[207] D. Marx. Graph colouring problems and their applications in scheduling. *Periodica Polytechnica Electrical Engineering*, 48(1):11–16, 2004.

[208] H. Meyniel. A new property of critical imperfect graphs and some consequences. *European Journal of Combinatorics*, 8:313–316, 1987.

[209] B. Monien and E. Speckenmeyer. Solving satisfiablility in less than 2^n steps. *Discrete Applied Mathematics*, 10:287–295, 1985.

[210] R. Mosca. Polynomial algorithms for the maximum stable set problem on particular classes of P_5-free graphs. *Information Processing Letters*, 61:137–143, 1997.

[211] R. Mosca. Stable sets in certain P_6-free graphs. *Discrete Applied Mathematics*, 92:177–191, 1999.

[212] R. Mosca. Some results on maximum stable sets in certain P_5-free graphs. *Discrete Applied Mathematics*, 132:175–183, 2004.

[213] R. Mosca. Some observations on maximum weight stable sets in certain P_5-free graphs. *European Journal of Operational Research*, 184:849–859, 2008.

[214] R. Mosca. Independent sets in $(P_6, \text{diamond})$-free graphs. *Discrete Mathematics and Theoretical Computer Science*, 11(1):125–140, 2009.

[215] H. Müller. Hamiltonian circuits in chordal bipartite graphs. *Discrete Mathematics*, 156:291–298, 1996.

[216] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Oxford University Press, 2006.

[217] R. Niedermeier and P. Rossmanith. Upper bounds for vertex cover further improved. In: *Proceedings of the 16th International Symposium on Theoretical Aspects of Computer Science (STACS 1999)*, volume 1563 of *Lecture Notes in Computer Science*, pages 561–570, Springer, 1999.

[218] C.H. Papadimitriou. Euclidean TSP is NP-complete. *Theoretical Computer Science*, 4:237–244, 1977.

[219] C.H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[220] C.H. Papadimitriou and M. Yannakakis. Optimization, approximation and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991.

[221] S. Parter. The use of linear graphs in Gauss elimination. *SIAM Review*, 3(2):119–130, 1961.

[222] K.R. Parthasarathy and G. Ravindra. The Strong Perfect Graph Conjecture is true for $K_{1,3}$-free graphs. *Journal of Combinatorial Theory, Series B*, 21:212–223, 1976.

[223] D. Paulusma and J.M.M. van Rooij. On partitioning a graph into two connected subgraphs. In: *Proceedings of the 20th International Symposium on Algorithms and Computation (ISAAC 2009)*, volume 5878 of *Lecture Notes in Computer Science*, pages 1215–1224, Springer, 2009.

[224] A. Pekeč and F.S. Roberts. The role assignment model nearly fits most social networks. *Mathematical Social Sciences*, 41:275–293, 2001.

[225] S. Poljak. A note on stable sets and colorings of graphs. *Commentationes Mathematicae Universitatis Carolinae*, 15:307–309, 1974.

[226] B. Randerath and I. Schiermeyer. 3-Colorability $\in \mathcal{P}$ for P_6-free graphs. *Discrete Applied Mathematics*, 136(2-3):299–313, 2004.

[227] B. Reed. ω, Δ, and χ. *Journal of Graph Theory*, 27:177–212, 1998.

[228] Y. Rieck and Y. Yamashita. Finite planar emulators for $K_{4,5}-4K_2$ and $K_{1,2,2,2}$ and Fellows' conjecture. *European Journal of Combinatorics*, 31(3):903–907, 2010.

[229] F.S. Roberts. Indifference graphs. In: *Proof Techniques in Graph Theory*, pages 139–146, Academic Press, New York, 1969.

[230] F.S. Roberts and L. Sheng. How hard is it to determine if a graph has a 2-role assignment? *Networks*, 37:67–73, 2001.

[231] N. Robertson and P.D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7:309–322, 1986.

[232] N. Robertson and P.D. Seymour. Graph minors. XIII. The disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63:65–110, 1995.

[233] N. Robertson and P.D. Seymour. Graph Minors. XX. Wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92:325–357, 2004.

[234] J.M. Robson. Algorithms for maximum independent sets. *Journal of Algorithms*, 7:425–440, 1986.

[235] J.M.M. van Rooij and H.L. Bodlaender. Design by measure and conquer – A faster exact algorithm for dominating set. In: *Proceedings of the 25th Annual Symposium on Theoretical Aspects of Computer Science (STACS 2008)*, volume 3404 of *Lecture Notes in Computer Science*, pages 657–668, Springer, 2005.

[236] D.J. Rose, R.E. Tarjan, and G.S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on Computing*, 5(2):266–283, 1976.

[237] N.D. Roussopoulos. A $\max\{m, n\}$ algorithm for determining the graph H from its line graph G. *Information Processing Letters*, 2:108–112, 1973.

[238] Z. Ryjáček. On a closure concept in claw-free graphs. *Journal of Combinatorial Theory, Series B*, 70:217–224, 1997.

[239] R.M. Sampaio and C. Linhares Sales. On the complexity of finding even pairs in planar perfect graphs. *Electronic Notes in Discrete Mathematics*, 7:186–189, 2001.

[240] C.R. Satyan and C. Pandu Rangan. The parity path problem on some subclasses of perfect graphs. *Discrete Applied Mathematics*, 68(3):293–302, 1996.

[241] N. Sbihi. Algorithme de recherche d'un stable de cardinalité maximum dans un graphe sans étoile. *Discrete Mathematics*, 29(1):53–76, 1980 (in French).

[242] I. Schiermeyer. Solving satisfiablity in less than $O(1.579^n)$ steps. *Selected papers from Computer Science Logic (CSL 1992)*, volume 702 of *Lecture Notes in Computer Science*, pages 379–394, Springer, 1992.

[243] I. Schiermeyer. Pure literal look ahead: an $\mathcal{O}^*(1.497^n)$ 3-Satisfiability algorithm. In: *Proceedings of the Workshop on the Satisfiability Problem*, pages 127–136, 1996.

[244] U. Schöning. A probabilistic algorithm for k-SAT and constraint satisfaction problems. In: *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1999)*, pages 410–414, IEEE Computer Society, 1999.

[245] U. Schöning. Algorithmics in exponential time. In: *Proceedings of the 22th International Symposium on Theoretical Aspects of Computer Science (STACS 2005)*, volume 3404 of *Lecture Notes in Computer Science*, pages 36–43, Springer, 2005.

[246] D. Seinsche. On a property of the class of n-colorable graphs. *Journal of Combinatorial Theory, Series B*, 16:191–193, 1974.

[247] R. Shamir, R. Sharan, and D. Tsur. Cluster graph modification problems. *Discrete Applied Mathematics*, 144(1-2):173–182, 2004.

[248] R. Sharan, A. Maron-Katz, and R. Shamir. CLICK and EXPANDER: a system for clustering and visualizing gene expression data. *Bioinformatics*, 19(14):1787–1799, 2003.

[249] L. Sheng. 2-Role assignments on triangulated graphs. *Theoretical Computer Science*, 304:201–214, 2003.

[250] S. Shrem, M. Stern, and M.C. Golumbic. Smallest odd holes in claw-free graphs. In: *Proceedings of the 35th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2009)*, volume 5911 of *Lecture Notes in Computer Science*, pages 329–340, Springer, 2009.

[251] J. Spinrad. *Efficient Graph Representations*. Fields Institute Monographs, volume 19, American Mathematical Society, Providence, Rhode Island, 2003.

[252] D.P. Sumner. Graphs with 1-factors. *Proceedings of the American Mathematical Society*, 42(1):8–12, 1974.

[253] D.P. Sumner. Dacey graphs. *Journal of the Australian Mathematical Society*, 18:492–502, 1974.

[254] R.E. Tarjan. Decomposition by clique separators. *Discrete Mathematics*, 55:221–232, 1985.

[255] R.E. Tarjan and A.E. Trojanowski. Finding a maximum independent set. *SIAM Journal on Computing*, 6:537–546, 1977.

[256] R.E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 13:566–579, 1984.

[257] C. Thomassen and B. Toft. Non-separating induced cycles in graphs. *Journal of Combinatorial Theory, Series B*, 31:199–224, 1981.

[258] N. Trotignon. *Graphes parfaits: Structure et algorithmes.* PhD Thesis, Université Joseph Fourier - Grenoble I, 2004 (in French).

[259] Y. Villanger, P. Heggernes, C. Paul, and J.A. Telle. Interval completion is fixed parameter tractable. *SIAM Journal on Computing*, 38(5):2007–2020, 2009.

[260] J.R. Walter. *Representations of rigid cycle graphs.* PhD thesis, Wayne State University, Detroit, 1972.

[261] M. Waterman and J.R. Griggs. Interval graphs and maps of DNA. *Bulletin of Mathematical Biology*, 48:189–195, 1986.

[262] S.H. Whitesides. A method for solving certain graph recognition and optimization problems, with applications to perfect graphs. *Annals of Discrete Mathematics*, 21:281–297, 1984.

[263] H. Whitney. Congruent graphs and the connectivity of graphs. *American Journal of Mathematics*, 54:150–168, 1932.

[264] G.J. Woeginger. Exact algorithms for NP-hard problems: A survey. *Combinatorial Optimization — Eureka, You Shrink!*, volume 2570 of *Lecture Notes in Computer Science*, pages 185–207, Springer, 2003.

[265] G.J. Woeginger. Open problems around exact algorithms. *Discrete Applied Mathematics*, 156(3):397–405, 2008.

[266] G.J. Woeginger and J. Sgall. The complexity of coloring graphs without long induced paths. *Acta Cybernetica*, 15(1):107–117, 2001.

[267] E.S. Wolk. The comparability graph of a tree. *Proceedings of the American Mathematical Society*, 13:789–795, 1962.

[268] E.S. Wolk. A note on "The comparability graph of a tree". *Proceedings of the American Mathematical Society*, 16:17–20, 1965.

[269] Z. Wu and R. Leahy. An optimal graph theoretic approach to data clustering: theory and its applications to image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(11):1101–1113, 1993.

[270] D. Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In: *Proceedings of the 38th ACM Symposium on Theory of Computing (STOC 2006)*, pages 671–680, 2006.