The Domatic Number Problem: Boolean Hierarchy Completeness and Exact Exponential-Time Algorithms

Dissertation
zur Erlangung des Doktorgrades der
Mathematisch-Naturwissenschaftlichen Fakultät
der Heinrich-Heine-Universität Düsseldorf

vorgelegt von

Tobias Riege

geboren in Haltern

Düsseldorf, im November 2006

Aus dem Institut für Informatik der
Heinrich-Heine-Universität Düsseldorf

Gedruckt mit der Genehmigung der
Mathematisch-Naturwissenschaftlichen Fakultät der
Heinrich-Heine-Universität Düsseldorf

Referent:	Prof. Dr. Jörg Rothe
Koreferent:	Prof. Dr. Egon Wanke
Tag der mündlichen Prüfung:	20.12.2006

Acknowledgments

I want to thank my thesis advisor Jörg Rothe for all of his support during the past four years. In the first place, I am deeply grateful to him for giving me the chance to be part of his research team. Without his great efforts, I would never have had the chance to work in the scientific community. Many inspiring and valuable discussions with him initiated fruitful ideas that led to the research presented in this thesis. I am very thankful to him for letting me have a share of his mathematical precision. In addition, his profound knowledge of the English language helped a great deal to spice up our articles.

Next I would like to thank all my coauthors, including Jörg Rothe, Gábor Erdélyi, Holger Spakowski, and Masaki Yamamoto. Thanks also to Gerd Wechsung for pointing out that `Exact-2-DNP` is coNP-complete and Dieter Kratsch for giving the hint to Lawler's algorithm for the colorability problem.

Many thanks go out to all the great colleagues at the Department of Computer Science of the Heinrich-Heine-University Düsseldorf. For being the boss: Jörg. For sharing the room with me: Gábor. For countless cigarettes: Thomas. For all the Mad-Tea-Parties after going to the mensa: Cristian, Christopher, Johanna, Evgenia, Stefan. For the technical support: Guido. You all will be missed.

My research was supported by the German Science Foundation (DFG) under grants RO 1202/9-1 and RO 1202/9-3.

My family always supported me during the last 29 years. Johannes, Gabriele, Christian, Benjamin, Anna, and Johannes jr.: Thanks for being there!

Last but not least, I would like to thank Egon Wanke for serving as a reviewer for this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The core of computer science is to find efficient ways to solve various problems with the help of computers. These problems arise in highly diverse fields of mathematics, computer science, physics, and so on. From a practical point of view, the challenge of computer science is to fit complex real-world objects into suitable representations for the computer, so that the data can then be processed in an automated way. In contrast, one mission of theoretical computer science—and in particular of complexity theory—is to categorize each single problem according to its structural properties, i.e., to determine how hard it is to actually come up with a valid solution. Here, we will focus on the second question and examine the hardness of interesting graph problems of practical importance.

The classification of problems according to their computational complexity is done via complexity classes. Each such class is defined by two major entities. First, we need an algorithmic model which processes the input data for the problem in a well-defined way. Based on this device, we can then define a complexity measure; problems hard to solve will use up more of this resource than their easy counterparts. Each complexity class thus defined contains all problems which can be solved by this machine within the limits of a specified bound on the complexity measure.

The most important resource is time.[1] How long will it take until our algorithmic device is finished with processing the input data? Another important question is what time bound will be called "efficient," or in other words, what time period from the start of the computation until the output of a solution will be acceptable for us. During the 1960's, Cobham [Cob64] and Edmonds [Edm65] came up with the idea to call all problems efficiently

[1] After all, time equals money and money is important.

solvable for which an algorithm produces a valid solution within polynomial time relative to the size of the input. The complexity class P was born, and with the work of Hartmanis and Stearns [HS65], the formal notions of complexity classes and measures were introduced. The multitape Turing machine was chosen as the algorithmic device, combining the simple, elegant, and beautiful presentation of computation with robustness when compared to other models. One particularly important complexity measure defined was the computation time of the Turing machine relative to the size of the input. Adapting the diagonalization techniques used by Turing, Gödel, Church, and others in recursive function theory (in particular, in order to prove the undecidability of the Halting Problem [Tur36] and other problems), Hartmanis and Stearns established the first true hierarchy of complexity classes [HS65]. This formal framework initiated a rich research stream on structural complexity theory, the study of the relationships between different complexity classes. For an overview of the exciting starting years of computational complexity and a comprehensive list of references to the first results in structural complexity, see [Ste90].

Due to the practical importance of the class P, it was attempted to prove membership in this fundamental class for various computational problems. Unfortunately, many interesting problems of practical importance do not seem to lie within the class P; their complex structure rendered all efforts to design efficient algorithms for them unsuccessful.[2] This dissatisfying situation triggered the research with the focus on a complexity class beyond P: the class of problems solvable in *non*deterministic polynomial time, NP. Many of the apparantly infeasible problems are contained in NP. They all share the characteristic that, given a potential solution to the problem, it can be verified in polynomial time whether this guessed solution indeed is a valid solution. A small example illustrates this concept.

Suppose we are given graph $G = (V, E)$, consisting of a set of vertices V and a set of edges E, where each edge connects two vertices of V. The task to legally color the vertices of G is a well-known and thoroughly studied graph-theoretical problem. A coloring is a mapping from the vertices to the positive integers, which represent distinct colors. A coloring is said to be legal if no two vertices sharing an edge are of the same color. In Figure 1.1, a legal coloring of a graph with five vertices is shown.

In the formal definition of the coloring problem, the input consists of a graph G and a positive integer k, and we need to determine if G can legally

[2] Garey and Johnson later called these problems "intractable" [GJ79].

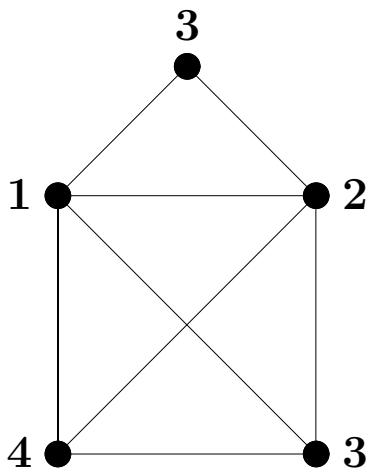

Figure 1.1: An example for the coloring problem

be colored using at most k colors.[3] It is easy to see that the colorability problem lies within the class NP. Given G and a coloring of at most k colors, it can efficiently be verified that any two vertices sharing an edge are mapped to different colors.

Clearly, P $\subseteq$ NP. This evidently leads to the question of whether there are sets that belong to NP, but *not* to P. This is the famous P $\stackrel{?}{=}$ NP question that has fascinated researchers all over the world for over 30 years. During the early 1970's, it was discovered that one could classify the hardest sets of a complexity class by the concept of reducibility. Given two sets A and B, a polynomial-time many-one reduction f is a transformation of all inputs $x \in \Sigma^*$ such that $x \in A$ if and only if $f(x) \in B$. If such a reduction between A and B exists, we write $A \leq_{\mathrm{m}}^{\mathrm{p}} B$. Neglecting the polynomial factor induced by reduction f, the set A is not harder to decide than the set B, since any algorithm for B straightforwardly yields a method to decide problem A in essentially the same time. If, for any complexity class $\mathcal{C}$, $C \leq_{\mathrm{m}}^{\mathrm{p}} B$ for all sets $C \in \mathcal{C}$, we call the set B "hard" (more precisely, $\leq_{\mathrm{m}}^{\mathrm{p}}$-hard) for the class $\mathcal{C}$. If simultaneously $B \in \mathcal{C}$, we call the set B "complete" (more precisely, $\leq_{\mathrm{m}}^{\mathrm{p}}$-complete) for $\mathcal{C}$. Relating these notions to the P $\stackrel{?}{=}$ NP question, if P $\neq$ NP then no NP-complete problem can be solved in polynomial time. On the other hand, if there is a (deterministic)

[3] We are concerned only with the decision version of the computational problem instead of the corresponding search variant. Note that for an NP-complete problem, the search version can efficiently be reduced to the decision version.

polynomial-time algorithm for any NP-complete problem, then P = NP.

A major breakthrough came with the discovery of the first natural NP-complete problem by Cook [Coo71]. We note that an analogous result was independently obtained by Levin [Lev73]. By a very clever proof, Cook showed how to reduce an arbitrary set in NP to the satisfiability problem, which asks for a given boolean formula φ if there exists a truth assignment satisfying φ. Based on Cook's results and the transitivity of the polynomial-time many-one reducibility, Karp showed dozens of problems NP-complete [Kar72]. Until today, literally thousands of problems have been classified as being NP-complete, including the colorability problem for all $k \geq 3$, see [Kar72]; as mentioned above, for no such problem can there exist a polynomial-time algorithm unless P = NP.

The domatic number problem is yet another important NP-complete problem that apparantly cannot be solved efficiently. This problem is the central topic of this thesis: Given any graph $G = (V, E)$ and a positive integer k, is it possible to partition the vertex set V into k vertex disjoint dominating sets? A dominating set D is a subset $D \subseteq V$ such that every vertex $v \in V - D$ has a neighbor in D. The decision problem k-DNP contains all graphs that can be partitioned into k disjoint dominating sets, and by $\delta(G)$ we denote the maximum number of vertex disjoint dominating sets of graph G.

The domatic number problem has been studied in various contexts. It was first defined by Cockayne and Hedetniemi [CH77]. In their paper, they answer the question of how many edges are needed for a graph with n vertices to contain $d = \delta(G)$ dominating sets. Their research was motivated by the fact that the domatic number problem is related to the problem of finding transmitting groups in a communication network. Suppose we are given a network of n cities which are connected via communication channels. A *transmitting group* is a subset of the cities that is able to communicate with every city in the network. Thus, it is nothing other than a dominating set, and the maximum number of disjoint transmitting groups is the domatic number of the graph modelled by the n cities.

Another real-world scenario where the domatic number problem arises is the allocation of resources in a computer network. Let a network of n facilities capable of storing information be given, where an edge between two nodes represents the fact that these nodes can share their information. If every vertex can store only a limited amount of information, there is a bound on the number of resources that can be supported. In the special case that every facility can only store one single resource, the maximum

number of resources that can be stored and accessed by every single node of the network equals the domatic number of the graph.

These two scenarios show that the domatic number is of great practical importance. However, for all $k \geq 3$ the set k-DNP was shown to be NP-complete [GJ79]. An elegant reduction from the colorability problem to 3-DNP is due to Kaplan and Shamir [KS94]. Their polynomial-time many-one reduction will play a crucial role in Theorem 3.1, one of the main results in this thesis. Kaplan and Shamir also prove that k-DNP is NP-complete when restricted to certain families of perfect graphs, including bipartite and chordal graphs. In contrast, k-DNP has been shown to be solvable in polynomial time for any strongly chordal graph G, where it holds that $\delta(G) = \mathit{min\text{-}deg}(G) + 1$,[4] see [Far84]. Another type of graphs for which the domatic number problem is efficiently solvable are the proper circular-arc graphs, whereas the problem remains NP-complete for general circular-arc graphs [Bon85]. Another important result for the domatic number problem is due to Feige et al. [FHKS03]. They give an $(1 + o(1)) \ln n$ approximation algorithm and additionally show that this might be the best bound possible, since an $(1 - \epsilon) \ln n$ approximation algorithm would imply that NP $\subseteq$ DTIME($n^{\log n \log n}$).

Coming back to structural complexity theory and leaving the already complex structure of NP and its sets, we advance to problems presumably lying beyond this complexity class. In their pathbreaking paper [MS72], Meyer and Stockmeyer analyzed natural problems with the intention to show the amazing computational hardness that can occur apart from the artificial sets that previously were constructed with diagonalization techniques by Hartmanis and Stearns [HS65]. Intriguingly, Meyer and Stockmeyer showed that any exponential-space computation can be reduced to the problem of determining whether two regular expressions with squaring are not equivalent. While this issue is appealing by itself, they also examined the problem of identifying boolean formulas for which no shorter, equivalent formula exists.[5] Let this set of boolean formulas be called MINIMAL. Meyer and Stockmeyer showed that the complement of this set, $\overline{\texttt{MINIMAL}}$, can be decided by an NP machine having access to an NP oracle,[6] so it is contained

[4] The term *min-deg*(G) denotes the minimum degree of graph G, i.e., the minimum number of neighbors taken over all vertices of G.

[5] Two boolean formulas φ_1 and φ_2 over the same variable set $X = \{x_1, x_2, \ldots, x_n\}$ are called *equivalent* if they evaluate to the same value for each of the 2^n different truth assignments.

[6] An informal description of an oracle Turing machine is given in Section 2.2.

in $\mathrm{NP}^{\mathrm{NP}}$. This suggests a hierarchy beyond NP that builds on nondeterministic polynomial-time reductions [MS72], and the polynomial hierarchy PH was born. As shown by Meyer and Stockmeyer [MS72], equality of any two distinct levels of the polynomial hierarchy implies a collapse of the entire hierarchy to this very level.

The exact complexity of `MINIMAL` is still unknown. Meyer and Stockmeyer did not provide a lower bound in their original paper [MS72]. It was later proven that `MINIMAL` is coNP-hard by E. Hemaspaandra and Wechsung [HW97], where the class coNP consists of all languages that are complements of sets in NP. This is still the best lower bound known until today. Consider a slight variant of the set `MINIMAL`: the set `MEE-DNF` contains all pairs $\langle \varphi, k \rangle$ of boolean formulas φ in disjunctive normal form and positive integers k such that there exists a boolean formula ϕ with at most k variables that is equivalent to φ. Umans showed that this problem is complete for Σ_2^p, the second level of the polynomial hierarchy [Uma98]. Extending the input to boolean formulas that need not to be in disjunctive normal form, define the problem `MEE`, which is trivially contained in Σ_2^p. This set was proven to be Θ_2^p-hard by E. Hemaspaandra and Wechsung, see [HW02]. The complexity class Θ_2^p can be characterized in many ways. It was first introduced by Papadimitriou and Zachos [PZ83] under the name $\mathrm{P}^{\mathrm{NP}[\mathcal{O}(\log)]}$, the class of sets decidable by a deterministic polynomial-time Turing machine being allowed to make a logarithmic number (with respect to the input size) of queries to an NP oracle set. One further characterization of Θ_2^p is given by the class $\mathrm{P}_{||}^{\mathrm{NP}}$, which contains all sets decidable by a deterministic polynomial-time Turing machine being allowed to make parallel (i.e., nonadaptive) queries to an NP oracle set. The result $\mathrm{P}^{\mathrm{NP}[\mathcal{O}(\log)]} = \mathrm{P}_{||}^{\mathrm{NP}}$ was independently obtained by L. Hemaspaandra [Hem87, Hem89] and Köbler, Schöning, and Wagner [KSW87].

Many problems have been proven to be complete for levels in the polynomial hierarchy. Section 3.4 of this thesis proves variants of the domatic number problem to be complete for $\mathrm{P}_{||}^{\mathrm{NP}}$. Wagner [Wag87] provided a set of tools to show completeness for this complexity class and the classes of the boolean hierarchy over NP, which will be discussed in the next paragraph. For further information about complete problems in the higher levels of the polynomial hierarchy, we refer to the two compendiums by Schaefer and Umans [SU02a, SU02b].

Another complexity class (and, based on it, an entire hierarchy of classes) was introduced to capture the complexity of problems that most likely cannot be decided by any NP Turing machine. To start with an example, the

set SAT-UNSAT contains all pairs (φ_1, φ_2) of boolean formulas such that φ_1 is satisfiable, whereas φ_2 is not. SAT-UNSAT is contained in the class DP, which was introduced by Papadimitriou and Yannakakis [PY84] as the class of differences of any two sets in NP.[7] The class DP can alternatively be formulated so as to contain all sets that are intersections of an NP and a coNP set. Papadimitriou and Yannakakis identified many problems that lie within DP, including critical and unique solution problems. Additionally they accomplished to prove some of them complete for DP. They noted that completeness proofs for DP—especially the ones for the critical problems—are quite hard to achieve. More DP-completeness results were obtained by Cai and Meyer for the minimal three uncolorability problem [CM87], and by Papadimitriou and Wolfe for minimal unsatisfiability and the traveling salesman facet problem [PW88].

When defining the complexity class DP in [PY84], it was evident to Papadimitriou and Yannakakis that it lies between NP and P^{NP}, the class of sets decidable by a deterministic polynomial-time Turing machine having access to an NP oracle set. The class P^{NP} is the second level of the polynomial hierarchy. Cai et al. generalized the class DP by defining the boolean hierarchy over NP [CGH$^+$88, CGH$^+$89]. The union of all complexity classes of the boolean hierarchy over NP is called BH(NP).[8] Similar to the polynomial hierarchy, it was shown that the boolean hierarchy over NP collapses if any two of its levels coincide. Wagner [Wag87] provided a set of conditions sufficient to prove completeness in each of the levels of BH(NP); moreover, he unified many different characterizations of the class $\mathrm{P}_{||}^{\mathrm{NP}}$ mentioned above, and introduced the name Θ_2^p for it [Wag90]. Note that $\mathrm{BH(NP)} \subseteq \Theta_2^p$. Kadin showed that a collapse of the boolean hierarchy over NP implies a collapse of the polynomial hierarchy to its third level [Kad88]. The currently strongest collapse result connecting the boolean hierarchy over NP with the polynomial hierarchy was independently achieved by E. and L. Hemaspaandra and Hempel [HHH98], and Reith and Wagner [RW01]. They show for each $k > 0$ and $i > 0$ that from the assumption $\mathrm{BH_k}(\Sigma_i^p) = \mathrm{coBH_k}(\Sigma_i^p)$, it follows that $\mathrm{PH} = \mathrm{BH_k}(\Sigma_i^p)\mathbf{\Delta}\mathrm{BH_{k-1}}(\Sigma_{i+1}^p)$, where $\mathbf{\Delta}$ symbolizes the complex symmetric difference of complexity classes. More relationships between these two hierarchies can be found in [CK96].

Last but not least, we will be dealing with exact and randomized algorithms for optimization problems running in exponential time. By "ex-

[7] In the original paper, the class DP was written D^{P}.

[8] Note that boolean hierarchies can be defined over arbitrary set rings, not only over the class NP.

act" we mean that the computation ends with a correct answer, unlike for approximation algorithms. The research on exponential-time algorithms has increased substantially during the last years. While at first it may seem not overly useful to search for algorithms running in exponential time, exponential-time algorithms that are better than the trivial exponential-time algorithm do have a very good motivation from a practical point of view. Consider an algorithm with a time bound of 2^n. It might be feasible to run this method for inputs of small size, and improving the time bound even further to, let's say, $\sqrt{2}^{\,n}$, one is able to run the advanced algorithm on inputs twice as large within the same time and still come up with a solution. Furthermore, the analysis of the methods obtained may have a huge impact when designing efficient algorithms for other problems. Interesting surveys on this subject can be found in [Sch05, Woe03]. For many more details on exponential-time algorithms, we refer to Section 5.1 in Chapter 5, as well as to the survey by Riege and Rothe [RR06c].

Coming to the notion of randomized algorithms, these probabilistic methods first occured during the 1970's when Solovay and Strassen [SS77] as well as Miller and Rabin [Mil76, Rab80] were driven by the search to find fast algorithms for detecting the primality of integers. Prime numbers are of great importance in a variety of practical applications, especially in cryptography. Since no efficient deterministic algorithms were known to construct large prime numbers, it was acceptable to use methods for this task with a one-sided error, i.e., there was no guarantee on the primality of the output. By successively repeating this procedure, the error rate can be made negligibly small, say below the probability that a hardware failure produces an erroneous output. Although the primality problem has later been proven to be solvable in deterministic polynomial time [AKS04], which is a milestone result in theoretical computer science, randomized primality tests are still more useful in practice due to their better time bounds. Randomized algorithms have also been constructed for NP-complete problems, for example the colorability [BE05] and the k-SAT problem [Sch99].

We will now give an overview of the structure of this thesis.

In Chapter 2, all relevant definitions for this thesis are provided. Most importantly, the notions of complexity class, polynomial-time many-one reducibility as well as the definition of the polynomial hierarchy and the boolean hierarchy over NP are given.

Chapter 3 introduces the domatic number problem, which is the core decision problem analyzed in this thesis. This is a classical graph problem that is one of the standard NP-complete problems, see [GJS76]. One of the

proofs of NP-hardness is witnessed by a clever polynomial-time many-one reduction from 3-COLOR to 3-DNP due to Kaplan and Shamir [KS94]. The properties of this reduction will be useful for Theorem 3.1, which is one of the main results in this thesis and proves the DP-hardness of k-DNP for all $k \geq 5$. The proof of this theorem and interesting open questions are discussed in Section 3.2. Even harder problems are studied in Section 3.3, when we analyze the complexity to determine if the domatic number of any graph equals one of k given integers. This problem is shown to be complete for the $2k$th level of the boolean hierarchy over NP. In Section 3.4, variants of the domatic number problem and their computational complexity are studied. It is shown that these variants are complete for the class $\mathrm{P}_{||}^{\mathrm{NP}}$. All these results have been published in [RR06b]; a preliminary version has appeared as [RR04].

Chapter 4 introduces a notation defined by Heggernes and Telle to characterize graph problems where the vertex set needs to be partitioned into generalized dominating sets [HT98]. These partitioning problems are described by two parameters, and after establishing the definitions in Section 4.1, the NP-completeness results of Heggernes and Telle for various parameters are listed in Section 4.2. Defining the exact versions of the generalized partitioning problems, completeness results for the class DP are obtained for three specific pairs of parameters in Section 4.3. Section 4.4 summarizes all results obtained in this thesis about the computational complexity of the exact versions of partitioning problems; they have been published in [RR06b] as well. In the same paper, the authors additionally show that exact versions of the conveyor flow shop problem are complete for levels in the boolean hierarchy over NP. This problem, which arises in real-world applications in the wholesale business, where warehouses are supplied with goods from a central storehouse, was introduced and intensely studied by Espelage and Wanke [EW00].

In Chapter 5, the last chapter of this thesis, exponential-time algorithms are presented which solve the domatic number problem exactly. Section 5.1 gives an overview of the diverse field of exponential-time algorithms. During the last few years, great progress has been made in the design and runtime analysis of these methods. The first algorithm solving the three domatic number problem in time below the trivial 3^n barrier, which is due to Riege and Rothe [RR05], is presented in Section 5.3. Further progress is made by adopting a technique dubbed "measure and conquer," which was studied intensely by Fomin, Kratsch, and Grandoni [FGK05a]. The technique goes back to an idea by Eppstein [Epp04]. Combining the result of Fomin et al.

on an algorithm listing all minimal dominating sets of a given graph with an algorithm for the satisfiability problem by Yamamoto [Yam05], an algorithm with the currently best bound to solve the special case of the three domatic number problem is developed in Section 5.4; this result has been published in [RRSY06a] and it was presented in [RRSY06b]. Chapter 5 concludes with Section 5.6, where one deterministic and two randomized algorithms for the domatic number problem—restricted to graphs with bounded degree—are constructed. These results can be found in [RR05].

Chapter 2

Notations

2.1 Sets and Languages

Denote the set of nonnegative integers by $\mathbb{N} = \{0, 1, 2, \ldots\}$ and denote the set of positive integers by $\mathbb{N}^+ = \{1, 2, \ldots\}$. Fix the alphabet $\Sigma = \{0, 1\}$. All our computations will be done over Σ^*, which is the set of strings over Σ with finite length. For any string $x \in \Sigma^*$ with length n, we denote by $|x|$ the *length of* x. A set $L \subseteq \Sigma^*$ over strings in Σ^* will be called *language* or *problem*. The *cardinality of* L is defined by $||L||$; it is the number of elements in the set L.

For any two languages A and B, define the following set operations:

- the *union of* A *and* B: $A \cup B = \{x \in \Sigma^* \mid x \in A \vee x \in B\}$,
- the *intersection of* A *and* B: $A \cap B = \{x \in \Sigma^* \mid x \in A \wedge x \in B\}$,
- the *complement of* A: $\overline{A} = \{x \in \Sigma^* \mid x \notin A\}$,

where $\vee$ denotes the logical `OR`-operation and $\wedge$ the logical `AND`-operation.

2.2 Machines and Reducibilities

The core of complexity theory is to determine the precise computational complexity of a given language L, that is, how hard it is for an algorithm to decide for a given input x if $x \in L$. First we have to settle the question what we mean by the term *algorithm*. An algorithm takes as input a string x over Σ, and by applying a fixed finite number of rules and methods, it either terminates by accepting x, rejecting x, or never halting at all. Optionally

the algorithm may output a string $y = f(x)$ depending on input x. Instead of deciding a language L, the algorithm computes a function in this case. Most of the time we will be speaking of deciding algorithms.

There are three characteristics which define the computational complexity of a given language L. First, one has to specify the *algorithmic model* which will be used to solve L. Our standard model of computation is the Turing machine (TM), which was named due to its inventor Alan Turing [Tur36]. The precise formal definition of the TM can be found in various textbooks about complexity theory, see [Pap94, HMU01, Rot05], we will only give a short description. A Turing machine M consists of an input tape, one or more working tapes, and a finite number of states. Each tape is divided into an infinite number of tape cells, which can each hold one symbol of the input alphabet. In addition there is a head attached to each tape that points on the tape cell that is currently being read. The tape cells on the working tape(s) can be overwritten. The heart of every Turing machine is its *finite transition table*, which maps its current state and the symbols being read on its tapes to another state, possibly overwriting the symbols being read on the working tapes. All heads can then move accordingly one position to the left, to the right, or not at all. A *configuration of* M is well-defined by the current state, the strings on the tapes, and the positions of the heads on the tapes. Given as input a string $x \in \Sigma^*$ over the finite input alphabet Σ, the computation of M on input x, in short $M(x)$, is a sequence of configurations starting with string $x = x_1 x_2 \dots x_n$ on the input tape and the read-only head on the first symbol x_1. All other tapes are empty at the beginning of the computation. The first state is the uniquely defined initial state z_0.

Coming back to the computational complexity of a given language L, the second characteristic that needs to be specified is the *acceptance mode* for the used algorithmic device, i.e., to precisely define what conditions have to occur during the computation such that the Turing machine accepts input $x \in \Sigma^*$. Here, we distinguish between *deterministic* and *nondeterministic* Turing machines (DTM, and NTM respectively). In the deterministic case, each configuration has at most one well-defined successor configuration, whereas the transition table of a nondeterministic Turing machine may map one configuration to more than one successor configuration. For a DTM M, we say that M *accepts input* x, if the computation of M on input x ends in an accepting configuration, i.e., reaches a well-defined accepting state z_y.

In contrast to the DTM, the computation of a NTM M on input x may branch into more than one successor configuration in every step. Therefore,

the computation of M on any input x is best described by the concept of the *computation tree*, where each node represents one configuration that is reachable from the initial configuration in a finite number of steps. The root node of the tree represents the initial configuration, and every node in the tree has as its child nodes its successor configurations. A *computation path of* M is a subset of the computation tree, where the subset includes the root node and a following sequence of configurations which correspond to a path in the computation tree. We say that M *accepts* x if at least one path in the computation tree of M ends in the accepting state z_y.

For any language L, a deterministic Turing machine M accepts L if each string x is accepted by M if and only if $x \in L$. Equivalently, we say that *set* L *is decided by* M. Analogously to the deterministic case, we say that the nondeterministic Turing machine M accepts language L if for each string $x \in L$, there exists at least one accepting path in the computation tree of M on input x.

Probability is yet another mode of acceptance for a TM that we will need in Section 5.6. Here, we will only briefly describe the acceptance mode of a probabilistic (randomized) Turing machine. During the computation, a probabilistic TM may take randomized choices, i.e., flip a coin (randomly choose between 0 and 1), and carry on with the computation depending on the outcome of the choice. The computation of a probabilistic TM on input x can then be seen as in the nondeterministic case; a computation tree reflects the randomized computation. Beginning with the initial configuration as the root node, the computation tree branches in each node (=configuration) where a random choice is made (in two or even more child nodes). Given a certain probability distribution on each random choice that can be made, the probability to reach a leaf (=end of computation) in the computation tree can then be calculated. A probabilistic TM M accepts input x if the probability to reach an accepting configuration exceeds a certain value, for example for $1/2$. For the complete formal definition of probabilistic computation on Turing machines, we refer the reader to [Gil77].

Another variant to this computational model is the concept of *oracle Turing machines*. Such a TM M, be it deterministic or nondeterministic, additionally receives a device called oracle, which consists of a set $L \in \Sigma^*$ and a special query tape. We mark that M may use L as an oracle set by writing M^L. During the computation of M^L on input x, machine M may enter a special marked state $z_?$. Suppose y is the current string on the query tape. In the next step of the computation, M enters state z_{yes} if it is $y \in L$, and z_{no} if $y \notin L$. The query tape is erased and the computation

continues as defined by the finite transition table of M. The crucial point is that deciding if y belongs to set L took exactly one step. Hence, using complex sets L as oracles, M may itself be able to decide significantly more complex sets than without the help of L.

Last but not least, the computational complexity of language L crucially depends on the complexity measure used. The two most basic measures in complexity theory are time and space, i.e., how many steps M needs to end the computation on input x, and how many tape cells are used during the course of the algorithm, respectively.

A DTM M is a *deterministic polynomial-time bound Turing machine*, or in short DPTM, if there exists a fixed polynomial $p \in \text{poly}$, such that for each input x, the computation halts after at most $p(|x|)$ steps. In the nondeterministic case, given a NTM M, every computation path in the computation tree of M on input x has to be of length at most $p(|x|)$ for M. Then we call M a *nondeterministic polynomial-time bound Turing machine*, or in short NPTM. In this thesis, we will only consider *worst-case complexity*, which is based on the definition of the $\mathcal{O}$-notation. For a given function $f : \mathbb{N} \to \mathbb{N}$, the function class $\mathcal{O}(f)$ contains all functions g which grow aymptotically slower than f, neglecting constant factors and finitely many exceptions:

$$\mathcal{O}(f) = \{g : \mathbb{N} \to \mathbb{N} \mid (\exists c > 0)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)[g(n) \leq c \cdot f(n)]\}.$$

In the worst-case complexity model, the function t which serves as a time or space bound needs to be sharp for all but a finite amount of inputs. The computation $M(x)$ is allowed to take at most $t(n)$ steps (tape cells) for *every* x with $|x| = n$.

Instead of using polynomials to bound the time (space) used in the computation of $M(x)$, other function families might serve as a limit to the number of steps (tape cells) of the computation. In the next section, we will define complexity classes (i.e., classes of sets) that can be solved with an algorithmic device given a certain acceptance mode and specifying a bound on the complexity measure. Finding an algorithm to decide a language L only indicates an upper bound on the computational complexity of L. To precisely capture how hard it is to solve L, that is, to provide a lower bound complexity class, we need the notion of reducibility.

Definition 2.1 *Let $\mathcal{FP}$ be the family of total functions mapping from Σ^* to Σ^* that can be computed in polynomial time with respect to the input size. For two sets A and B, we say that A* is polynomial-time many-one

reducible to B *if there exists a polynomial-time function* $f \in \mathcal{FP}$ *such that the equivalence*

$$x \in A \iff f(x) \in B \tag{2.1}$$

holds for all $x \in \Sigma^*$. *If such a reduction exists, we write* $A \leq_{\mathrm{m}}^{\mathrm{p}} B$.

The above definition is extremely useful when comparing the computational complexity of two different problems A and B. Given an algorithm for set B and a polynomial-time many-one reduction f certifying that $A \leq_{\mathrm{m}}^{\mathrm{p}} B$, we can decide language A in the following way. On input x, first compute f on input x. Afterwards, decide if $f(x) \in B$ through Equation (2.1). This way we decided A, adding only a polynomial amount of effort to the computational complexity to solve B. This explains the phrase *set A is reduced to B*. Given that A represents a computational complexity above deterministic polynomial time, reduction f is a certificate for a lower bound on the complexity of B. We can constitute that problem B is at least as hard to solve than A, since any algorithm solving B straightforwardly leads to another (though slightly more complicated) method deciding A. The next section will introduce the concept of complexity classes, where computational problems of similar structure are categorized.

2.3 Complexity Classes and Hierarchies

In complexity theory, complexity classes and hierarchies are used to classify the various and diverse problems which occur and need to be solved in computer science. The class P contains all languages which can be recognized by a DTM in polynomial-time. In 1965, Edmonds was the first to characterize P as the class containing problems that can be solved in "feasible" time [Edm65]. The nondeterministic analog of P is the class NP, which contains every set which can be decided by a NTM working in polynomial time. Clearly, it holds that P $\subseteq$ NP, for each deterministic TM is a special case of a nondeterministic TM. This leads to the most significant question of complexity theory, whether P equals NP or if they differ, i.e., if P is a proper subset of NP. This issue has not been solved until today. Another class is given by coNP, the set of languages L where the complement set $\overline{L}$ is a member of NP.

Both, P and NP, are *complexity classes*, that is, both are sets of languages. For a complexity class $\mathcal{C}$ and a set $B \subseteq \Sigma^*$, we now define the notions of hardness and completeness.

Definition 2.2 *Let $\mathcal{C}$ be a complexity class, and let language B be given. The set B is said to be* $\leq_{\mathrm{m}}^{\mathrm{p}}$-hard for the class $\mathcal{C}$, *if for each $A \in \mathcal{C}$, it holds that $A \leq_{\mathrm{m}}^{\mathrm{p}} B$. We simply write B* is $\mathcal{C}$-hard, *since all reductions in this thesis are polynomial-time many-one reductions $\leq_{\mathrm{m}}^{\mathrm{p}}$. The set B is called* $\leq_{\mathrm{m}}^{\mathrm{p}}$-complete for the class $\mathcal{C}$, *if the following two conditions hold:*

- *$B \in \mathcal{C}$, and*
- *B is $\leq_{\mathrm{m}}^{\mathrm{p}}$-hard for $\mathcal{C}$.*

Again we simply write B is $\mathcal{C}$-complete. *The complexity class $\mathcal{C}$ is* closed under the reducibility $\leq_{\mathrm{m}}^{\mathrm{p}}$ *if and only if for any two sets A and B, if $B \in \mathcal{C}$ and $A \leq_{\mathrm{m}}^{\mathrm{p}} B$, it follows $A \in \mathcal{C}$. In short, we write* class $\mathcal{C}$ is $\leq_{\mathrm{m}}^{\mathrm{p}}$-closed.

We have found a simple—yet ingenious—way to prove if two complexity classes are equal to one another. Suppose that for two given complexity classes $\mathcal{C}$ and $\mathcal{D}$, it is known that $\mathcal{C} \subseteq \mathcal{D}$. For example, this applies to the two classes P and NP defined above. Assuming that $\mathcal{C}$ is closed under $\leq_{\mathrm{m}}^{\mathrm{p}}$-reductions, it now suffices to show that B is contained in $\mathcal{C}$ for merely one $\mathcal{D}$-complete problem to prove that $\mathcal{C} = \mathcal{D}$! In the case of the P vs NP question, it would suffice to construct a polynomial-time algorithm for one NP-complete problem to solve this matter once and for all.

Many problems have been categorized to be solvable in deterministic polynomial time. Many other natural problems arising in the real world have been shown to be NP-complete. Still, during more than 30 years of research, starting with the first results on NP-completeness by Cook and Karp [Coo71, Kar72], no algorithm working in polynomial time has been found for an NP-complete problem, which might be an indicator that P does not equal NP.

For the following complexity classes to be defined, we need to introduce the notion of computation relative to a complexity class $\mathcal{C}$. Remember the informal description of an oracle Turing machine in Section 2.2.

Definition 2.3 *The complexity class P^A consists of all sets L which can be decided by a DPTM with access to an oracle $A \in \Sigma^*$. The class NP^A contains all languages L which can be decided by an NPTM with access to oracle A. Generalizing the notion of computation relative to oracles for any class $\mathcal{C}$, we define the complexity classes*

$$\mathrm{P}^{\mathcal{C}} = \bigcup_{A \in \mathcal{C}} \mathrm{P}^A \text{ and } \mathrm{NP}^{\mathcal{C}} = \bigcup_{A \in \mathcal{C}} \mathrm{NP}^A.$$

Given a family of functions $\mathcal{F}$, the complexity class $\mathrm{P}^{A[\mathcal{F}]}$ ($\mathrm{NP}^{A[\mathcal{F}]}$) consists of all languages decidable by a DPTM (an NPTM) having access to the oracle set A, where the number of queries is bound by a function $f \in \mathcal{F}$, i.e., at most $f(|x|)$ queries can be made to oracle A on any input x.

Apparently, Turing machines are able to decide a wider class of problems when having access to an oracle. While still no efficient algorithm could be found for the hardest problems in NP, the NP-complete sets, there are even languages which are beyond NP and seemingly much harder to solve. To capture the computational complexity of sets beyond NP, we will define two hierarchies of complexity classes. At first, we mention the polynomial hierarchy, which has been introduced by Meyer and Stockmeyer [MS72]. They studied the problem of finding an equivalent expression φ' of size less than or equal to an integer k for a given boolean formula φ. Two boolean formulas φ and φ' over the same variable set X are called equivalent if each truth assignment $t : X \to \{0,1\}$ evaluates to the same value for φ and φ'; see Definition 2.9 for the notion of boolean formulas. Meyer and Stockmeyer noticed that this problem is coNP-hard, but they failed to prove membership in coNP. For they found this language solvable with a coNP machine having access to an NP oracle, they introduced a hierarchy of classes which capture the complexity of problems which seem to lie beyond NP and coNP.

Definition 2.4 *The* polynomial hierarchy *is inductively defined as follows:*

$$\begin{aligned}
\Delta_0^p &= \Sigma_0^p = \Pi_0^p = \mathrm{P}, \\
\Delta_{k+1}^p &= \mathrm{P}^{\Sigma_k^p}, \\
\Sigma_{k+1}^p &= \mathrm{NP}^{\Sigma_k^p}, \\
\Pi_{k+1}^p &= \mathrm{coNP}^{\Sigma_k^p}, \\
\mathrm{PH} &= \bigcup_{k \geq 0} \Sigma_k^p.
\end{aligned}$$

As a special case, we define the class $\Theta_2^p = \mathrm{P}^{\mathrm{NP}[\log]}$, which contains all problems decidable by an oracle DPTM having access to a set $A \in \mathrm{NP}$, where at most a logarithmic number of queries can be made.

Figure 2.1, which is taken from [RR06a], illustrates the inclusion structure of the polynomial hierarchy. A line between two complexity classes denotes the fact that the lower class is a subset of the higher class.

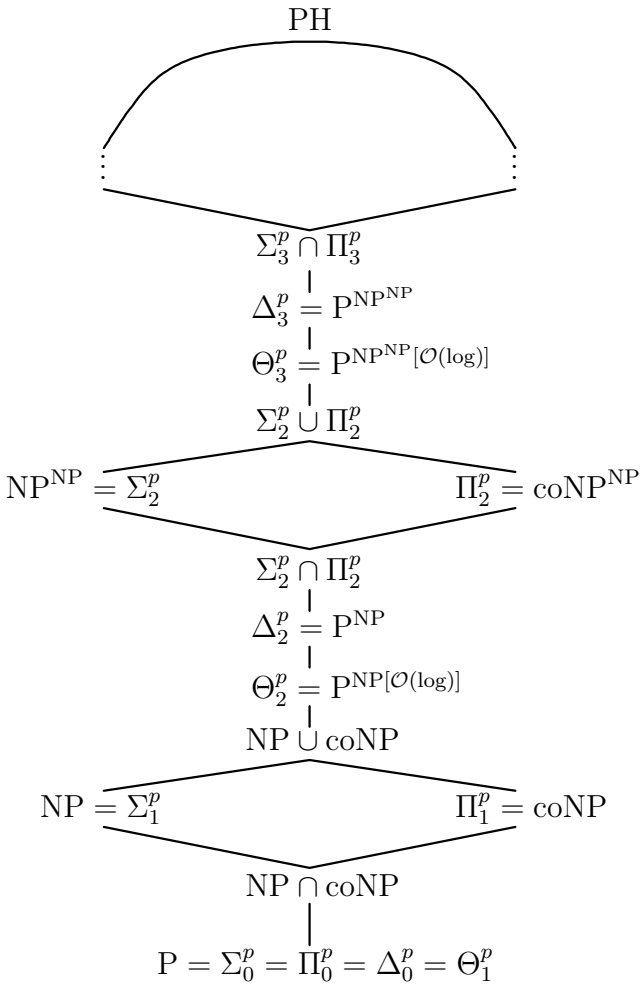

Figure 2.1: The polynomial hierarchy

Hierarchies of complexity classes describe the subtle differences between computational complexities of various problems that occur in computer science. Another such hierarchy is the *boolean hierarchy over* NP, which lies within the polynomial hierarchy and hence represents an even finer measure of differences in computational complexity. This hierarchy was introduced by Cai et al. [CGH$^+$88, CGH$^+$89]. Before defining the boolean hierarchy over NP, which plays a major role in this thesis, we need to introduce some operations on complexity classes.

The *complex intersection* and the *complex union* of complexity classes $\mathcal{C}$

and $\mathcal{D}$ is defined as

$$\begin{aligned} \mathcal{C} \wedge \mathcal{D} &= \{A \cap B \mid A \in \mathcal{C} \text{ and } B \in \mathcal{D}\}, \\ \mathcal{C} \vee \mathcal{D} &= \{A \cup B \mid A \in \mathcal{C} \text{ and } B \in \mathcal{D}\}. \end{aligned}$$

This definition is not to be mistaken by the intersection and union between sets A and B, see Section 2.1. We are now ready to define another hierarchy of complexity classes.

Definition 2.5 *The* boolean hierarchy over NP *is inductively defined as follows:*

$$\begin{aligned} \mathrm{BH}_0(\mathrm{NP}) &= \mathrm{P}, \\ \mathrm{BH}_1(\mathrm{NP}) &= \mathrm{NP}, \\ \mathrm{BH}_2(\mathrm{NP}) &= \mathrm{NP} \wedge \mathrm{coNP}, \\ \mathrm{BH}_k(\mathrm{NP}) &= \mathrm{BH}_{k-2}(\mathrm{NP}) \vee \mathrm{BH}_2(\mathrm{NP}) \textit{ for } k \geq 3, \textit{ and} \\ \mathrm{BH}(\mathrm{NP}) &= \bigcup_{k \geq 1} \mathrm{BH}_k(\mathrm{NP}). \end{aligned}$$

Figure 2.2, which is taken from [RR06a], illustrates the inclusion structure of the boolean hierarchy over NP. Note that BH(NP) lies within the class $\mathrm{P}^{\mathrm{NP}[\mathcal{O}(\log)]}$, which fits in between the first and second level of the polynomial hierarchy, see Figure 2.1.

For the second level $k = 2$, the class NP $\wedge$ coNP is also referred to by the name DP, the *difference of any* NP *sets*. It can be rewritten as

$$\mathrm{DP} = \{A - B \mid A, B \in \mathrm{NP}\}.$$

Many interesting problems lie within the class DP. Here, we will list a few of them:

- *Unique problems:* Given problem A, is it true that there exists *exactly one* solution to this problem?
- *Critical problems:* Given problem A with input x, is it true that there exists a solution to x, but removing one property of x leads to an instance without a solution?
- *Exact problems:* Given an optimization problem with parameter $p(x)$ for input x and a positive integer k, is it true that $p(x)$ exactly equals k?

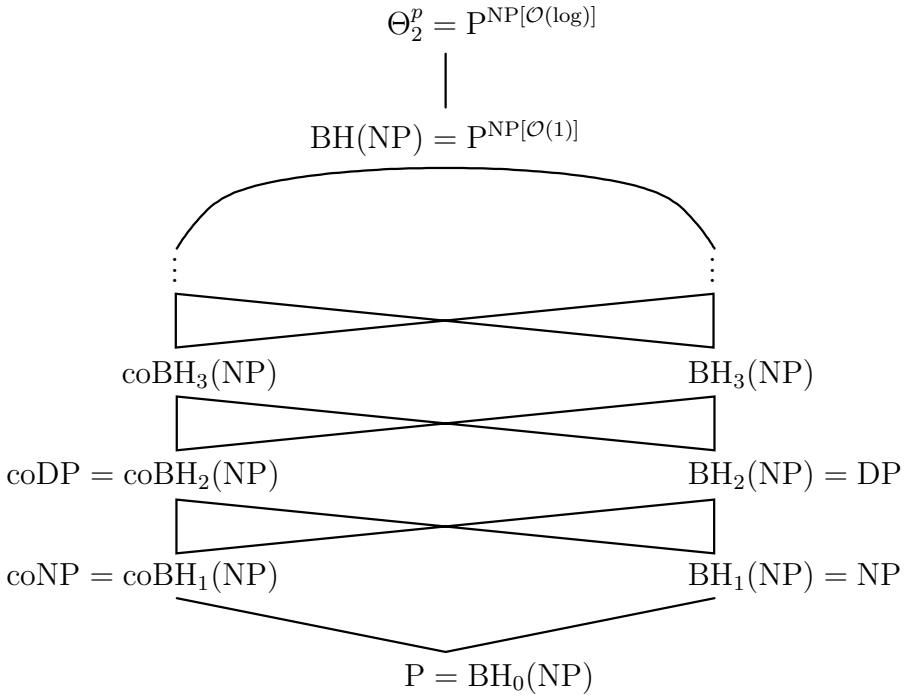

Figure 2.2: The boolean hierarchy over NP

The last item will be investigated for a couple of optimization problems, i.e., maximum or minimum problems, in Chapters 3 and 4. For a survey about many diverse complete problems in the boolean hierarchy over NP, we refer to [RR06a].

2.4 Boolean Formulas and Graphs

Many problems proven to be NP-complete are derived from questions concerning graphs or boolean formulas as input. In fact, the first problem shown to be NP-complete was the satisfiabilty problem. This major breakthrough was achieved by Cook [Coo71], and independently by Levin [Lev73]. The definitions to follow in this section will provide the basis to the decision problems described in Section 2.5.

Definition 2.6 *A* boolean formula *in conjunctive normal form (CNF) is a pair $\varphi = \varphi(X, C)$, where $X = \{x_1, x_2, \ldots, x_n\}$ is a set of boolean variables, and C is a conjunction of clauses over literals from X. A* literal *is either a variable $x_i \in X$ or its negation $\overline{x_i}$, $1 \leq i \leq n$. A* truth assignment *is a*

mapping $t : X \to \{0, 1\}$ *from the variable set* X *to the values* 0 *(false) and* 1 *(true). Truth assignment* t *satisfies literal* x_i *if* $t(x_i) = 1$*, and it satisfies the negated literal* $\overline{x_i}$ *if* $t(x_i) = 0$*. We say that truth assignment* t *satisfies the boolean formula* $\varphi(X, C)$ *if every clause in* C *contains at least one literal that is satisfied by* t*. A boolean formula* $\varphi = \varphi(X, C)$ *is said to be in* k*-CNF if each clause in* C *contains at most* k *literals.*

Next we introduce some graph-theoretical notation. In this thesis, only simple, undirected graphs without self-loops are considered. That is, there is no edge of the form $\{u, u\}$ and at most one edge $\{u, v\}$ for each pair of vertices $u, v \in V$.

Definition 2.7 *A* graph $G = (V, E)$ *consists of* $V = \{v_1, v_2, \ldots, v_n\}$*, its set of* vertices*, and* $E = \{\{v_i, v_j\} \mid 1 \leq i < j \leq n\}$*, its set of* edges*, where each edge connects two vertices of* V*. Two vertices sharing an edge are called* adjacent*. For any vertex* $v \in V$*, the* open neighborhood $N(v)$ *includes all vertices adjacent to* v*, that is* $N(v) = \{u \in V \mid \{u, v\} \in E\}$*. The* closed neighborhood *of* v *in* G *is defined by* $N[v] = N(v) \cup \{v\}$*. We call* $v \in V$ *an* isolated vertex *if* $N[v] = \{v\}$*, i.e.,* v *is not connected to any other vertex in* V*. The* degree of vertex v in G *is denoted by* $deg_G(v)$*. If* G *is clear from the context, we omit the subscript and simply write* $deg(v)$*. The* maximum degree of G *is defined by* $max\text{-}deg(G) = \max_{v \in V} deg(v)$*, and the* minimum degree of G *is defined by* $min\text{-}deg(G) = \min_{v \in V} deg(v)$*, respectively. We call a graph* k-regular *if and only if* $deg(v) = k$ *for each vertex* $v \in V$*.*

For some of the results to follow, we will need to define basic operations that can be applied to graphs.

Definition 2.8 *For two given graphs* $G_1 = (V_1, E_1)$ *and* $G_2 = (V_2, E_2)$ *with disjoint vertex sets, we define the* disjoint union *by* $G_1 \cup G_2 = (V, E)$ *with* $V = V_1 \cup V_2$ *and* $E = E_1 \cup E_2$*. The* join operation on G_1 and G_2 *is denoted by* $G_1 \oplus G_2 = (V, E)$ *with*

$$\begin{aligned} V &= V_1 \cup V_2, \text{ and} \\ E &= E_1 \cup E_2 \cup \{\{u, v\} \mid u \in V_1 \text{ and } v \in V_2\}. \end{aligned}$$

For any subset V' *of the vertex set* V*, the* subgraph induced by V' *is defined by* $G[V'] = (V', E')$ *with edge set* $E' = \{\{u, v\} \in E \mid u \in V' \wedge v \in V'\}$*. In the* complement graph of G*, called* $\overline{G}$*, each pair of vertices* $u, v \in V$ *is connected by an edge if and only if they were not adjacent in* G*. Thus, it is* $\overline{G} = (V, E')$ *with edge set* $E' = \{\{u, v\} \mid \{u, v\} \notin E\}$*.*

Given these basic definitions, we are now able to define interesting problems that contain formulas or graphs which properties are subject to various different (and especially computationally hard to recognize) restrictions.

2.5 Decision Problems and Their Variants

A large portion of the NP-complete sets are derived from problems on boolean formulas and graphs. In applications arising in the real world, especially the family of graph problems appears quite often. However, we start by defining one of the classic NP-complete problems and its variants.

Definition 2.9 *For a given boolean formula $\varphi = \varphi(X, C)$, the* satisfiability problem `SAT` *asks whether there exists a satisfying truth assignment t with $t : X \to \{0, 1\}$ for φ. For any $k \geq 2$, define k-*`SAT` *to be the set containing all satisfiable formulas φ with at most k literals in each clause $c \in C$. Additionally, define the* not-all-equal satisfiability problem `NAE-3-SAT` *as the set of boolean formulas $\varphi = \varphi(X, C)$ such that all clauses contain exactly three literals, and for which there exists a satisfying truth assignment t such that in each clause $c \in C$, not all literals are mapped to the value* 1 *(true) under t. In other words, each clause must contain at least one true literal, and each clause must contain at least one false literal under assignment t. The set* 1-3-`SAT` *consists of those boolean formulas $\varphi = \varphi(X, C)$ in 3-CNF for which there exists a truth assignment t such that each clause $c \in C$ is comprised of* exactly one *literal that is satisfied under assignment t.*

The satisfiability problem was the first language proven to be complete for NP by Cook [Coo71]. He also showed that k-`SAT` is NP-complete for all $k \geq 3$. In Contrast, the set 2-`SAT` is decidable in polynomial time, Papadimitriou constructed an $\mathcal{O}(n^2)$ algorithm for it [Pap91]. Actually, the set 2-`SAT` is complete for the complexity class NL, see [JLL76]. The class NL includes all sets that are decidable by a nondeterministic Turing machine in logarithmic space. Note that NL $\subseteq$ P. It is not known whether this is a proper inclusion, i.e., if it holds that NL $\subset$ P. The restriction `NAE-3-SAT` of the satisfiability problem was proven to be NP-complete by Schaefer [Sch78], who also showed that 1-3-`SAT` is $\leq_{\mathrm{m}}^{\mathrm{p}}$-complete for NP. This problem even remains NP-complete when the input is restricted to formulas containing only positive literals in all clauses.

Definition 2.10 *For a graph $G = (V, E)$, an* independent set *I is a subset of the vertex set V such that no two vertices of I are adjacent. An indepen-*

dent set I of G with maximum cardinality is called maximum independent set*. In contrast, the set I is said to be a* maximal independent set*, if there exists no superset $I' \supset I$ such that I' is an independent set. Note that a maximum independent set is always maximal, whereas the converse need not to be true.*

For any positive integer $k \in \mathbb{N}$, define the *maximum independent set problem* `INDSET` to be the set containing all graphs $G = (V, E)$ with a maximum independent set $I \subseteq V$ of cardinality $||I|| \geq k$. This classic graph problem was among the 20 examples proven to be NP-complete by Karp [Kar72]. Actually, he constructed a reduction from `SAT` to `CLIQUE`, but it is easily verified that for any graph $G = (V, E)$, each set $I \subseteq V$ is a maximum independent set of G if and only if I is a clique in the complement graph $\overline{G}$. A clique is a subset V' of the vertex set such that for any two vertices $u, v \in V'$, it holds that $\{u, v\} \in E$. Based on the definition of independent set, another interesting challenge is to partition the vertex set V of a graph G into the maximum number of such independent sets.

Definition 2.11 *Let graph $G = (V, E)$ be given. A* k-coloring *of G is a mapping $f : V \to \{1, 2, \ldots, k\}$. The values from 1 to k are also called* color classes*. A k-coloring is called* legal *if no two adjacent vertices are mapped to the same color class. Graph G is said to be k-colorable if there exists a legal k-coloring for G. Define for any graph $G = (V, E)$ the* chromatic number $\chi(G)$ *as the minimum number k such that G is k-colorable. Equivalently, a graph G is k-colorable if its vertex set V can be partitioned into k independent sets.*

Coloring of a graph G is a minimization problem, as any graph G with a legal k-coloring simultaneously is $k+1$-colorable. For any positive integer k, define k-`COLOR` to be the problem to decide whether a given graph G is colorable with k colors:

$$k\text{-}\texttt{COLOR} = \{G \mid G \text{ is a graph and } \chi(G) \leq k\}.$$

The set k-`COLOR` was proven to be complete for the class NP for all $k \geq 3$ by Karp [Kar72]. It lies within P for $k = 2$, since a graph is colorable with two colors if and only if it is bipartite, which can be recognized in polynomial time.[1]

[1] A graph $G = (V, E)$ is called *bipartite* if its vertex set V can be partitioned into two disjoint sets V_1 and V_2 such that each edge has one endpoint in V_1 and one endpoint in V_2.

Definition 2.12 *For any graph $G = (V, E)$, a subset $D \subseteq V$ of the vertex set is called* dominating set *if every vertex $u \in V - D$ is adjacent to at least one vertex $v \in D$. In other words, a set D is dominating if $N[D] = V$. A* minimum dominating set *D of G is a dominating set of G with the smallest cardinality, i.e., there exists no dominating set D' of G with $||D'|| < ||D||$. For a dominating set D of G, we say that* vertex *$v \in V$* is dominated by *D if it holds $N[v] \cap D \neq \emptyset$. Note that each vertex $v \in V$ dominates itself by definition. In contrast to a minimum dominating set, a* minimal dominating set *D of G is a dominating set $D \subseteq V$ such that no proper subset $D' \subset D$ is a dominating set of G. A minimum dominating set is always a minimal dominating set, whereas the converse need not to be true.*

Analogously to the question of colorability for any graph G, where the property (independence) of a subset of the vertex set of G induces a partitioning problem (k-COLOR), the dominating sets directly lead to the partitioning domatic number problem.

Definition 2.13 *Define for graph $G = (V, E)$ the* domatic number *$\delta(G)$ to be the maximum number of vertex disjoint dominating sets of G. For a positive integer $k \in \mathbb{N}$, the decision problem k-DNP contains all graphs G with $\delta(G) \geq k$:*

$$k\text{-DNP} = \{G \mid G \text{ is a graph and } \delta(G) \geq k\}.$$

It is easy to see that $\delta(G) \leq \textit{min-deg}(G) + 1$, since each vertex can only be dominated by itself and each of its neighbors. Also, we note that every graph $G = (V, E)$ without isolated vertices has a domatic number of at least two. For example, construct two vertex disjoint dominating sets by choosing one maximal independent set I and the remaining vertices in the set $V - I$.

Fact 2.1 *The set* 2-DNP *lies within* P.

The domatic number problem is NP-complete for all $k \geq 3$ as noted by Garey and Johnson [GJ79]. Lemma 3.1 in the next chapter gives as a constructive proof for this claim a polynomial-time many-one reduction that will also be used to prove the DP-completeness of the exact version of the domatic number problem in Section 3.2.

Chapter 3

The Domatic Number Problem

3.1 Overview

The domatic number problem is a graph partitioning problem. The underlying dominating sets which build up this partition have been studied intensely over the last decades. The minimum dominating set problem, in short DOMSET, asks for a given graph G and a positive integer k if there exists a dominating set of G of size at most k. It was noted to be complete for NP by Garey and Johnson [GJ79]. The proof on the NP-hardness of DOMSET, which is witnessed by a reduction from 3-SAT, is described in full detail in [HHS98b]. More than a thousand papers have been published on the topic of domination in graphs. Most results have been merged in the two excellent surveys by Haynes, Hedetniemi, and Slater [HHS98b, HHS98a]. Still, results on the partitioning domatic number problem are scarce. In this thesis, we will try to solve diverse questions regarding the domatic number problem to gain a better understanding of this natural problem arising in real-world scenarios.

The domatic number problem k-DNP was noted to be NP-complete by Garey and Johnson [GJ79], notwithstanding that they do not give an explicit polynomial-time many-one reduction to prove NP-hardness. The following result due to Kaplan and Shamir [KS94] catches up on this flaw. Their reduction from 3-COLOR proves that k-DNP is NP-complete for $k \geq 3$. They additionally show that the problem remains NP-hard even when the input is restricted to split graphs.

Lemma 3.1 *The problem k-DNP is* NP*-complete for $k \geq 3$.*

Proof. At first it will be proven that k-DNP is NP-complete for $k = 3$.

Extending the proof for all $k > 3$ is straightforward. The set k-DNP is included in NP by the following simple algorithm. Nondeterministically put each vertex $v \in V$ in one of the sets $V_1, V_2, \ldots, V_k$ of the partition. Afterwards, verify for $1 \leq i \leq k$ that each V_i is a dominating set of G. This can be done in polynomial time.

To prove that 3-DNP is NP-hard, we will present a reduction from the NP-complete set 3-COLOR by Kaplan and Shamir [KS94]. Assume without loss of generality that the input graph G contains no isolated vertices and is not colorable with two colors. The polynomial-time many-one reduction g constructs a graph $G' = (V', E')$ from the input graph $G = (V, E)$ as follows:

$$\begin{aligned} V' &= V \cup \bigcup_{\{v_i, v_j\} \in E} \{u_{i,j}\}, \\ E' &= \bigcup_{\{v_i, v_j\} \in E} (\{v_i, u_{i,j}\} \cup \{u_{i,j}, v_j\}) \bigcup_{v_i, v_j \in V, 1 \leq i < j \leq n} \{v_i, v_j\}. \end{aligned}$$

The vertex set V' includes all vertices of V, and for each edge $\{v_i, v_j\} \in E$, one vertex $u_{i,j}$ is added to V'. The vertex $u_{i,j}$ is connected to v_i and v_j via two edges. Finally, all original vertices from V are connected pairwise to form a clique of size n.

We will prove the following implications:

$$G \in \text{3-COLOR} \implies \delta(G') = 3, \tag{3.1}$$

$$G \notin \text{3-COLOR} \implies \delta(G') = 2. \tag{3.2}$$

Then, G is colorable with three colors if and only if G' can be partitioned into three disjoint dominating sets.

Notice that $2 \leq \delta(G') \leq 3$ follows from the two facts that G contains no isolated vertices and the degree of each induced edge vertix $u_{i,j}$ equals two. Now suppose $\chi(G) = 3$, and let $f : V \to \{1, 2, 3\}$ be the mapping for this three-coloring. Define the three nonempty sets

$$V_i = \{v \in V \mid f(v) = i\}.$$

We will construct three dominating sets D_1, D_2, and D_3 for graph G' as follows. For $1 \leq i \leq 3$, define the sets

$$D_i = V_i \cup \{u_{i,j} \mid v_i \notin V_i \text{ and } v_j \notin V_i\}.$$

Clearly, since each of the three color classes V_i is nonempty and the original vertices in V form a clique in G', each D_i, $1 \leq i \leq 3$, dominates V in G'.

Every triangle $\{v_i, u_{i,j}, v_j\}$ contains exactly one member of each set D_i, therefore the set $\{u_{i,j} \mid \{v_i, v_j\} \in E\}$ is dominated three times as well. It follows that $\delta(G) = 3$, and implication (3.1) is proven.

To show that implication (3.2) holds, assume that it is $\delta(G') = 3$ and let D_1, D_2, and D_3 be the three disjoint dominating sets of G'. Then, a legal three-coloring for the vertices $v \in V$ of G is given by $f(v) = i$ for $v \in D_i$. Note that two adjacent vertices $v_i, v_j \in V$ will not receive the same color, since the vertices of the triangle $\{v_i, u_{i,j}, v_j\}$ must belong to three different dominating sets.

To see that k-DNP is NP-complete for $k \geq 4$, a polynomial-time many-one reduction f from k-DNP to $(k+1)$-DNP is given by adding to any graph $G = (V, E)$ an additional vertex u, and connecting u to every vertex $v \in V$. For the generated graph $G' = G \oplus \{u\}$ it holds that $\delta(G') = \delta(G) + 1$. ∎

Note that the resulting graph $G' = g(G)$ is a split graph, i.e., its vertex set V' can be partitioned into a clique C and an independent set I. Here, it is $C = V$ and $I = \{u_{v_i,v_j} \mid \{v_i, v_j\} \in E\}$.

3.2 Exact Version of the Domatic Number Problem

Starting from an NP-complete problem A, one can define numerous variants of problem A with different computational complexities. For example we may restrict the input to problem A to certain subsets. On the one hand, the problem might be easier to solve, as is the case with the domatic number problem when only strongly chordal graphs are considered as input [Far84]. On the other hand, the complexity might stay the same, as is the case with determining the domatic number of chordal graphs [KS94].

Another way to change a problem is to make restrictions to the solution space. For instance, we want to be able identify all boolean formulas that are satisfiable but have only one unique solution, i.e., there exists only one distinct satisfying truth assignment. This set is called UNIQUE-SAT. Another interesting problem is given by the set of all graphs G that cannot be colored legally with three colors, but removing any vertex causes G to become three-colorable. This problem is known by the name MINIMAL-3-UNCOLOR.

Problems of the kind that the input has (has not) a certain property, but deleting or adding any element of the input removes (adds) the property

are called *critical problems*. Critical and unique solution problems are good candidates for DP-completeness, since they are easily shown to be contained in DP. But it is much more difficult to produce a proof to show the DP-hardness of critical problems. As was already noted by Papadimitriou and Yannakakis: "This difficulty seems to reflect the extremely delicate and deep structure of critical problems—too delicate to sustain any of the known reduction methods" [PY84].

Getting back to the two DP-problems above, the precise complexity of the set `UNIQUE-SAT` is still unknown until today, whilst DP-completeness of the problem `MINIMAL-3-UNCOLOR` has been proven by Cai and Meyer [CM87].

The focus of this thesis lies on *exact* optimization problems, that means that we are interested in the exact value of a certain property. In contrast to that, the question for NP-complete optimization problems usually contains a one-sided bound. For instance, in the case of the decision version of the domatic number problem it is asked whether the domatic number of a given graph and a positive integer k is *at least* as high as k. Next we will define the exact domatic number problem.

Definition 3.1 *Let M_k be a set of k positive integers. Define the* exact domatic number problem *by*

$$\texttt{Exact-}M_k\texttt{-DNP} = \{G \mid G \text{ is a graph and } \delta(G) \in M_k\}.$$

When it is $k = 1$ and $M_1 = \{t\}$, we simply write `Exact-`t`-DNP` *instead of* `Exact-`$\{t\}$`-DNP`.

The question is how hard it is to solve the exact domatic number problem, i.e., for which of the various number of complexity classes it has to be classified. In the case of `Exact-`t`-DNP`, one might suppose that this problem is NP-complete. Actually, reduction f from Lemma 3.1 proves that this problem is NP-hard, but what about the upper complexity bound of the set `Exact-`t`-DNP`? For a given graph G, its domatic number number $\delta(G)$ has to be determined precisely. A naive nondeterministic algorithm working in polynomial time is able to guess a partition into t vertex disjoint dominating sets, therefore the algorithm is able to decide if $\delta(G) \geq t$. But to show that it holds $\delta(G) < t+1$, *all* possible partitions into $t+1$ dominating sets have to be taken into account, and it is not clear how to achieve this with a nondeterministic Turing machine in polynomial time.

Looking back at Chapter 2, Definition 2.5 of the class DP is the solution to precisely determine the complexity of the exact domatic number problem,

and in fact of many other exact optimization problems in general. Recall that DP is defined as the class of sets which can be written as the intersection of two sets $A \in$ NP and $B \in$ coNP. Define the exact domatic number problem as Exact-k-DNP $= A \cap \overline{B}$ by the sets

$$\begin{aligned} A &= \{G \mid G \text{ is a graph and } \delta(G) \geq t\}, \\ B &= \{G \mid G \text{ is a graph and } \delta(G) \geq t+1\}. \end{aligned}$$

Clearly, it holds that $A \in$ NP and $B \in$ NP. The same argument can be used to prove membership in the higher levels of the boolean hierarchy over NP for the problems Exact-M_k-DNP.

Fact 3.1 *For any positive integer t, it is* Exact-t-DNP $\in$ DP. *For the general case $k \geq 1$, it is* Exact-M_k-DNP $\in \mathrm{BH_{2k}(NP)}$.

It leaves to show that this classification is "sharp," i.e., that the exact domatic number problem Exact-M_k-DNP is hard for $\mathrm{BH_{2k}(NP)}$. Wagner provided a set of conditions sufficient to proof hardness in the levels of the boolean hierarchy over NP [Wag87]. The next lemma states one of those conditions which will be usefull to prove the upcoming Theorem 3.1.

Lemma 3.2 *Let A be some* NP*-complete problem, let B be an arbitrary problem, and let $k \geq 1$ be fixed. If there exists a polynomial-time computable function f such that the equivalence*

$$||\{i \mid x_i \in A\}|| \text{ is odd} \iff f(x_1, x_2, \ldots, x_{2k}) \in B \tag{3.3}$$

is true for all strings $x_1, x_2, \ldots, x_{2k} \in \Sigma^*$ satisfying $x_{j+1} \in A$ implies $x_j \in A$ for each j with $1 \leq j \leq 2k$, then B is $\mathrm{BH_{2k}(NP)}$*-hard.*

For the special case $k = 1$, it suffices to verify the equivalence

$$(x_1 \in A \wedge x_2 \notin A) \iff f(x_1, x_2) \in B \tag{3.4}$$

for some function $f \in \mathcal{FP}$ to prove the DP-hardness of the set B.

With the help of Lemma 3.2, various exact optimization problems were proven complete for classes contained the boolean hierarchy over NP. For example, the exact versions of the independent set problem and the colorability problem—when defined analogously to Definition 3.1—have been shown to be complete for $\mathrm{BH_{2k}(NP)}$ in [Wag87]. Interestingly in the same paper, it was proven that the problem Exact-t-COLOR is DP-complete for

all values $t \geq 7$. Note that for the case $t = 3$, it is Exact-3-COLOR $= A \cap \overline{B}$ with

$$\begin{aligned} A &= \{G \mid G \text{ is a graph and } \chi(G) \leq 3\}, \\ B &= \{G \mid G \text{ is a graph and } \chi(G) \leq 2\}. \end{aligned}$$

It is $A \in$ NP, whereas the set B lies in P, as every bipartite graph is colorable with two colors, and bipartiteness can be recognized in polynomial time. Hence, Exact-3-COLOR is contained in NP and cannot be complete for DP unless the boolean hierarchy over NP collapses to its second level. In [Wag87], the question was raised how small the number t can be chosen such that Exact-t-COLOR remains DP-complete. In particular, what is the complexity of Exact-t-COLOR for $t \in \{4, 5, 6\}$? The question was solved by Rothe in [Rot03] with Lemma 3.2 and by combining two reductions to the problem 3-COLOR, more precisely the standard reduction f from 3-SAT to 3-COLOR by Garey and Johnson [GJS76] and a clever reduction g from Guruswami and Khanna [GK00] from INDSET to 3-COLOR with the nice property that

$$\begin{aligned} G \in \text{INDSET} &\iff \chi(g(G)) = 3, \\ G \notin \text{INDSET} &\iff \chi(g(G)) = 5. \end{aligned}$$

Rothe obtained the result that Exact-t-COLOR is DP-complete for all values $t \geq 4$ [Rot03]. We will state his result in Subsection 4.3.2, where additionally Rothe's results concerning the more generalized versions of exact colorability problems will be discussed.

The following Theorem is due to Riege and Rothe [RR06b]. It determines the precise complexity of Exact-t-DNP for all $t \geq 5$.

Theorem 3.1 *The problem* Exact-t-DNP *is* DP*-complete for* $t \geq 5$.

Proof. According to Fact 3.1, the set Exact-t-DNP is contained in the class DP $=$ NP $\wedge$ coNP for all $t \in \mathbb{N}$.

It suffices to prove the DP-hardness of Exact-t-DNP for $t = 5$, see also the remark at the end of the proof of Lemma 3.1 as how to extend DP-hardness to all values $t > 5$. We will make use of Lemma 3.2 with $k = 1$, 3-COLOR as the NP-complete set A and Exact-5-DNP being set B of this lemma. We will construct a polynomial-time many-one reduction f satisfying Equation (3.4) as follows.

Fix any two graphs G_1 and G_2 and without loss of generality, assume that both graphs do not contain isolated vertices and are not colorable with two

colors. Also, if $G_2 \in$ 3-COLOR, then so is G_1. At first, we apply reduction g of Lemma 3.1 to both graphs G_1 and G_2 to generate two graphs $H_1 = g(G_1)$ and $H_2 = g(G_2)$. The final step in our reduction f will be to construct a graph $H = f(G_1, G_2)$ such that

$$\delta(H) = \delta(H_1) + \delta(H_2). \tag{3.5}$$

This will be the hardest task to accomplish, and Equation (3.5) will straightforwardly lead to the equivalence

$$\begin{aligned} & G_1 \in \text{3-COLOR} \text{ and } G_2 \notin \text{3-COLOR} \\ \iff & \delta(H_1) = 3 \text{ and } \delta(H_2) = 2 \\ \iff & \delta(H) = \delta(H_1) + \delta(H_2) = 5 \\ \iff & f(G_1, G_2) = H \in \text{Exact-5-DNP}, \end{aligned}$$

which according to Lemma 3.2 proves the DP-hardness of Exact-5-DNP.

It remains to prove Equation (3.5). Recall that the polynomial-time many-one reduction g from Lemma 3.1 maps graph $G = (V, E)$ to a graph $H = g(G)$ such that each edge $e = \{v_i, v_j\}$ induces a triangle by adding vertex $u_{i,j}$ and connecting it to both endpoints of edge e. Additionally all original vertices in the vertex set V of G are connected such that they form a clique. Now fix two triangles $T_1 = \{v_q, u_{q,r}, v_r\}$ in $H_1 = g(G_1)$ and $T_2 = \{v_s, u_{s,t}, v_t\}$ in $H_2 = g(G_2)$. Insert six new vertices $a_1, a_2, \ldots, a_6$ and connect them to the triangles via the gadget shown in Figure 3.1. Each pair of triangles from H_1 and H_2 is connected by one disjoint copy of the gadget in Figure 3.1, which completes the description of reduction f. Note that f is polynomial-time computable. Figure 3.1 is taken from [RR06b].

There are three cases to distinguish where we will have to prove the claim $\delta(H) = \delta(H_1) + \delta(H_2)$ of Equation (3.5). In each case it holds that $\delta(H) \leq 6$, since each of the inserted gadget vertices is adjacent to exactly five vertices. Let $D_1, D_2, \ldots D_{\delta(H_1)}$ be the dominating sets of H_1 and $D_{\delta(H_1)+1}, D_{\delta(H_1)+2}, \ldots, D_{\delta(H_1)+\delta(H_2)}$ be the dominating sets of H_2.

Case 1: $[\boldsymbol{\delta(H_1) = \delta(H_2) = 3}]$ As mentioned above, it is $\delta(H) \leq 6$. We will construct six vertex disjoint dominating sets $\hat{D}_1, \hat{D}_2, \ldots, \hat{D}_6$ of graph H to prove $\delta(H) = 6$. Consider one dominating set D_j of H_1 with $1 \leq j \leq 3$. We will extend this set so as to yield a dominating set for the entire graph H. Fix a triangle $T_1 = \{v_q, u_{q,r}, v_r\}$ in $H_1 = g(G_1)$. Exactly one of the three vertices of triangle T_1 belongs to each of the three dominating sets. Suppose for D_1, it is $D_1 \cap T_1 = u_{q,r}$.

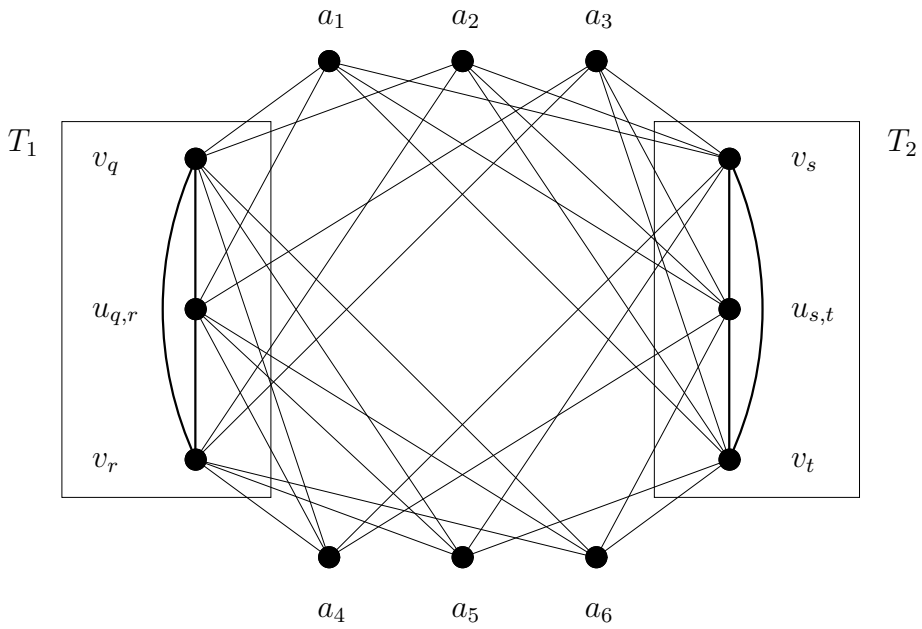

Figure 3.1: Gadget connecting two triangles T_1 and T_2

All other cases are treated analogously. Now consider one triangle $T_2 = \{v_s, u_{s,t}, v_t\}$ of $H_2 = g(G_2)$ and the gadget connecting T_1 and T_2. Let's call the additional vertices $a_1^{T_2}, a_2^{T_2}, \ldots, a_6^{T_2}$. Exactly one of the additional vertices is not adjacent to $u_{q,r}$, this is $a_2^{T_2}$, see Figure 3.1. We will add this vertex to D_1, so define

$$\hat{D}_1 = D_1 \cap \{a_2^{T_2} \mid T_2 \text{ is a triangle of } H_2\}.$$

This set is a dominating set for graph H. The graph H_1 was dominated by D_1 beforehand. Every gadget vertex a_i is dominated by $\hat{D}_1$ since the only vertex not adjacent to $u_{q,r}$ is added to $\hat{D}_1$. And finally every vertex of H_2 is contained in a triangle T_2. Therefore, as vertex $a_2^{T_2}$ is adjacent to all vertices of the triangle T_2, graph H_2 is dominated by the set $\hat{D}_1$ as well.

With the same argument, the dominating sets D_j of H_2,$4 \leq j \leq 6$, can be extended to dominating sets $\hat{D}_j$ for graph H. The sets $\hat{D}_j$ are pairwise vertex disjoint for $1 \leq j \leq 6$ by construction, hence it holds that $\delta(H) = \delta(H_1) + \delta(H_2) = 6$.

Case 2: $[\boldsymbol{\delta(H_1) = 3}$ **and** $\boldsymbol{\delta(H_2) = 2}]$ With the same argument as in the

first case, the dominating sets $D_1, D_2, \ldots, D_5$ of H_1 and H_2 can be extended to dominating sets $\hat{D}_1, \hat{D}_2, \ldots, \hat{D}_5$ of H. So $5 \leq \delta(H) \leq 6$. Assume that $\delta(H) = 6$, and let $E_1, E_2, \ldots, E_6$ be the six vertex disjoint dominating sets for H. Fix a triangle $T_1 = \{v_q, u_{q,r}, v_r\}$ of H_1. Look at Figure 3.1 with gadget vertex a_4, which has a degree of five and is adjacent to all three vertices of T_1. This implicates that the vertices of T_1 pairwise belong to different dominating sets, for example $v_q \in E_1$, $u_{q,r} \in E_2$, and $v_r \in E_3$. All other cases can be treated analogously.

Fixing a triangle $T_2 = \{v_s, u_{s,t}, v_t\}$ of H_2 and by a symmetric argument, the vertices of T_2 must be members of three different dominating sets. None of these three sets can be E_1, E_2, or E_3, as then one of the gadget vertices $a_1, a_2, \ldots, a_6$ would have two neighbors of the same dominating set, therefore contradicting the assumption that $\delta(H) = 6$. So the only way (except renaming the dominating sets) to assign the vertices of T_2 to the dominating sets E_i is $v_s \in E_4$, $u_{s,t} \in E_5$, and $v_t \in E_6$. This is true for each triangle T_2 of H_2, hence each of the three vertex disjoint sets E_4, E_5, and E_6 are dominating sets for H_2. This is a contradiction to $\delta(H_2) = 2$ and we can conclude that $\delta(H) = \delta(H_1) + \delta(H_2) = 5$.

Case 3: $[\boldsymbol{\delta(H_1) = \delta(H_2) = 2}]$ Just as in the first two cases, the four dominating sets D_1, D_2, D_3, and D_4 can be extended to vertex disjoint dominating sets $\hat{D}_1, \hat{D}_2, \hat{D}_3$, and $\hat{D}_4$ of the entire graph H. And analogously to Case 2, it can be shown that $\delta(H) = 6$ cannot hold. It remains to prove $\delta(H) \neq 5$. Suppose otherwise, so the vertices of H can be partitioned into five dominating sets $E_1, E_2, \ldots, E_5$. For a contradiction, fix again the triangles $T_1 = \{v_q, u_{q,r}, v_r\}$ of H_1 and $T_2 = \{v_s, u_{s,t}, v_t\}$ of H_2 as in Figure 3.1.

First we show that neither T_1 nor T_2 can have two vertices belonging to the same dominating set. Assume that for T_1, the vertices v_q and v_r belong to the same set, say E_1, and $u_{q,r}$ is in the set E_2. All other cases can be treated analogously. None of the vertices in triangle T_2 can belong to set E_1 or E_2, as at least one of the vertices a_4, a_5, and a_6 would then be adjacent to four vertices from E_1 and E_2, leaving only two more choices for the vertices in the closed neighborhood not yet added to a dominating set. This contradicts our assumption that $\delta(H) = 5$. Also, the vertices in T_2 have to belong to three different dominating sets, otherwise vertex a_2 which is adjacent to $v_q \in E_1$ and $v_r \in E_1$ cannot be dominated by five dominating sets. It follows

that $||V(T_2) \cap E_3|| = 1$, $||V(T_2) \cap E_4|| = 1$, and $||V(T_2) \cap E_5|| = 1$. This is true for each of the triangles T_2 of H_2, therefore we found a vertex disjoint partition of H_2 into three dominating sets. This is a contradiction to $\delta(H_2) = 2$. Hence, it is $\delta(H) = \delta(H_1) + \delta(H_2) = 4$.

The fourth case "$\delta(H_1) = 2$ and $\delta(H_2) = 3$" cannot occur since we have restricted the input such that from $G_2 \in$ 3-COLOR, it follows that $G_1 \in$ 3-COLOR. It is $\delta(H) = \delta(H_1) + \delta(H_2)$ and thus we have proven Equation (3.5) and hereby Theorem 3.1.

∎

The set Exact-2-DNP is coNP-complete since for any graph G the property $\delta(G) \geq 2$ can be checked in polynomial time, see Fact 2.1. The proof of coNP-hardness is given by reduction f from Lemma 3.1. Therefore, the set Exact-2-DNP cannot be DP-hard unless the boolean hierarchy over NP collapses to its first level.

The exact complexity of Exact-t-DNP is still an open question for the two cases $t = 3$ and $t = 4$.

Corollary 3.1 *The problem* Exact-2-DNP *cannot be* DP*-complete unless the boolean hierarchy over* NP *collapses to its second level.*

It is still an open question if the set Exact-t-DNP is DP-complete for the cases $t = 3$ and $t = 4$. In the case of the colorability problem, the precise cutoff point between NP- and DP-completeness was resolved by Rothe [Rot03], as Exact-3-COLOR is NP-complete and Exact-t-COLOR is DP-complete for all $t \geq 4$, see Theorem 4.3. To close the gap in the case of the domatic number problem in a similar way, two issues would still need to be resolved. First, as was the case with the colorability problem, a polynomial-time many-one reduction h to the domatic number problem needs to be constructed with the property that

$$\begin{aligned} a \in A &\implies \delta(h(a)) = 4, \\ a \notin A &\implies \delta(h(a)) = 2, \end{aligned}$$

where A is some arbitrary NP-complete set. Second, some way to construct a graph H with the property $\delta(H) = \max\{\delta(H_1), \delta(H_2)\}$ has to be found. Let the reduction accomplishing this task be denoted by f. Then, given two instances a_1 and a_2 of A with the property that if $a_2 \in A$, so is a_1, we could

use reduction g from Lemma 3.1 and the hypothetic reduction h mentioned above so that

$$\begin{aligned} & a_1 \in A \text{ and } a_2 \notin A \\ \iff & \delta(g(a_1)) = 3 \text{ and } \delta(h(a_2)) = 2 \\ \iff & \delta(f(H_1, H_2)) = \max\{\delta(H_1), \delta(H_2)\} = 3 \\ & \text{with } H_1 = g(a_1) \text{ and } H_2 = h(a_2) \\ \iff & f(H_1, H_2) = H \in \texttt{Exact-3-DNP}. \end{aligned}$$

In Theorem 3.1, the graph operation needed to show the completeness of Exact-5-DNP for the class DP was to add the domatic number of the two graphs H_1 and H_2. This was relatively easy to accomplish by the triangle structure of the generated graphs H_1 and H_2. This might not be the case when combining two different reduction methods. Note that addition and taking the minimum of chromatic numbers $\chi(G_1)$ and $\chi(G_2)$ of two given graphs G_1 and G_2 is achieved by the basic graph operations join $G_1 \oplus G_2$ and disjoint union $G_1 \cup G_2$. The exact computational complexity of Exact-3-DNP and Exact-4-DNP therefore remains an interesting open question.

3.3 The Higher Levels of BH(NP)

The last section focused on Exact-M_k-DNP, the exact domatic number problem for the special case $k = 1$. In the following, we will generalize the result from Theorem 3.1 for all $k \geq 1$.

Theorem 3.2 *For fixed $k \geq 1$, the problem* Exact-M_k-DNP *is complete for* $\mathrm{BH_{2k}(NP)}$ *for the set* $M_k = \{4k + 1, 4k + 3, \ldots, 6k - 1\}$.

Proof. Similar to the proof for membership of Exact-t-DNP in DP, divide the problem Exact-M_k-DNP into k subproblems

$$\texttt{Exact-}M_k\texttt{-DNP} = \bigcup_{t \in M_k} \texttt{Exact-}t\texttt{-DNP},$$

which can then be rewritten as

$$\texttt{Exact-}t\texttt{-DNP} = \{G \mid \delta(G) \geq t\} \cap \{G \mid \delta(G) < t + 1\}.$$

For any given graph $G = (V, E)$, the set $\{G \mid \delta(G) \geq t\}$ is a member of NP by an algorithm which nondeterministically branches into every possible

partition of vertex set V into t sets and then verifying for each guess, if each of the t sets is a dominating set of G. The second set is clearly contained in coNP, and as it is

$$\mathrm{BH_{2k}(NP)} = \bigvee_{i=1}^{k} (\mathrm{NP} \wedge \mathrm{coNP})$$

by definition, the set Exact-M_k-DNP is contained in $\mathrm{BH_{2k}(NP)}$.

It leaves to proof that the set Exact-M_k-DNP is hard for $\mathrm{BH_{2k}(NP)}$. This proof is a straightforward generalization of the proof of Theorem 3.1, where the DP-hardness of Exact-5-DNP was shown. We will make use of Lemma 3.2 with the NP-complete problem 3-COLOR being set A and the problem Exact-M_k-DNP representing set B. Let the graphs $G_1, G_2, \ldots, G_{2k}$ be given with the property that $G_{i+1} \in$ 3-COLOR implies $G_i \in$ 3-COLOR for each i with $1 \leq i < 2k$. Again we may assume without loss of generality that all graphs G_i, $1 \leq i \leq 2k$, do not contain isolated vertices, and that $3 \leq \chi(G_i) \leq 4$, since two-colorable graphs can be recognized in polynomial time. For each i with $1 \leq i \leq 2k$, we apply reduction g from Lemma 3.1 to obtain $2k$ graphs $H_i = f(G_i)$. Since Equations (3.1) and (3.2) are satisfied, it follows for each i with $1 \leq i < 2k$ that $\delta(H_{i+1}) = 3$ implies $\delta(H_i) = 3$.

Remember the construction from Theorem 3.1, where we added six gadget vertices a_i for each pair of triangles T_1 and T_2 from the two graphs H_1 and H_2. We extend this construction by adding $6k$ additional gadget vertices $a_1, a_2, \ldots, a_{6k}$ for each sequence of triangles $T_1, T_2, \ldots, T_{2k}$, where each T_i belongs to graph H_i for $1 \leq i \leq 2k$. Associate for each i with $1 \leq i \leq 2k$ the three gadget vertices $a_{1+3(i-1)}$, $a_{2+3(i-1)}$, and $a_{3+3(i-1)}$ with the triangle T_i. For $1 \leq j \leq 2k$ and $j \neq i$ connect triangle T_i to triangle T_j via the three gadget vertices $a_{1+3(i-1)}$, $a_{2+3(i-1)}$, and $a_{3+3(i-1)}$ with the same pattern as T_1 is connected to T_2 in Figure 3.1 via the gadget vertices a_1, a_2, and a_3. This completes the construction, and we call the resulting graph $H = f(H_1, H_2, \ldots, H_{2k})$. Note that f is computable in polynomial time.

For each a_i with $1 \leq i \leq 6k$ it is $deg(a_i) = 6k - 1$, since it is connected to every vertex in the triangles T_j, $1 \leq j \leq 2k$ except one. Hence, it holds that $\delta(H) \leq 6k$. With an analogous argument to the one in Theorem 3.1, it can be proven that $\delta(H) = \sum_{i=1}^{2k} \delta(H_i)$. Due to the fact that this argument is rather lengthy when it is extended to the general case of Exact-M_k-DNP, we omit the proof that is presented in full detail in [RR06b]. Thus, we can conclude directly that it holds

$$
\begin{aligned}
&||\{i \mid G_i \in \text{3-COLOR}\}|| \text{ is odd} \\
&\iff (\exists i : 1 \le i \le k) \left[\begin{array}{l} \chi(G_1) = \cdots = \chi(G_{2i-1}) = 3 \text{ and} \\ \chi(G_{2i}) = \cdots = \chi(G_{2k}) = 4 \end{array} \right] \\
&\iff (\exists i : 1 \le i \le k) \left[\begin{array}{l} \delta(H_1) = \cdots = \delta(H_{2i-1}) = 3 \text{ and} \\ \delta(H_{2i}) = \cdots = \delta(H_{2k}) = 2 \end{array} \right] \\
&\iff (\exists i : 1 \le i \le k) \left[\begin{array}{ll} \delta(H) & = \sum_{j=1}^{2k} \delta(H_j) \\ & = 3(2i-1) + 2(2k-2i+1) \end{array} \right] \\
&\iff (\exists i : 1 \le i \le k) \; [\; \delta(H) = 4k + 2i - 1 \;] \\
&\iff \delta(H) \in \{4k+1, 4k+3, \ldots, 6k-1\} \\
&\iff f(G_1, G_2, \ldots, G_{2k}) = H \in \text{Exact-}M_k\text{-DNP}.
\end{aligned}
$$

Equation (3.3) from Lemma 3.2 is satisfied and thus Exact-M_k-DNP is complete for $\mathrm{BH_{2k}(NP)}$.

∎

Note that the case $k = 1$ resembles Theorem 3.1, where it was shown that Exact-5-DNP is complete for the class $\mathrm{DP} = \mathrm{BH_2(NP)}$.

3.4 Harder Questions on Domination

In Section 3.1, it was shown that t-DNP is NP-complete for $t \ge 3$. In Section 3.2, it was shown that Exact-t-DNP is DP-complete for $t \ge 5$, and rising even higher in the boolean hierarchy over NP, it was proven that the set Exact-M_k-DNP is complete for $\mathrm{BH_{2k}(NP)}$. If the boolean hierarchy over NP does not collapse, i.e., if for each level i with $i \ge 0$ it is $\mathrm{BH_i(NP)}$ properly contained in $\mathrm{BH_{i+1}(NP)}$, every follow-up problem was a little bit harder to solve than the previous one. This section deals with even harder questions concerned about the domatic number of graphs. The following definition introduces problems that stand slightly higher above BH(NP), the boolean closure over NP.

Definition 3.2 *We define the following variants of the domatic number problem:*

$$
\begin{aligned}
\text{DNP} - \text{Odd} &= \{G \mid G \textit{ is a graph such that } \delta(G) \textit{ is odd}\}, \\
\text{DNP} - \text{Equ} &= \{(G, H) \mid G \textit{ and } H \textit{ are graphs such that } \delta(G) = \delta(H)\}, \\
\text{DNP} - \text{Geq} &= \{(G, H) \mid G \textit{ and } H \textit{ are graphs such that } \delta(G) \ge \delta(H)\}.
\end{aligned}
$$

The following Theorem is due to Wagner [Wag87]. It gives a sufficient condition proving the $\mathrm{P}_{||}^{\mathrm{NP}}$-hardness of a given problem B. The given condition is similar to the one given in Lemma 3.2, except that the value for the integer k is not fixed.

Lemma 3.3 *Let A be some* NP*-complete problem, and B be an arbitrary problem. If there exists a polynomial-time computable function f such that the equivalence*

$$||\{i \mid x_i \in A\}|| \text{ is odd} \quad \Leftrightarrow \quad f(x_1, x_2, \ldots, x_{2k}) \in B \tag{3.6}$$

is true for each $k \geq 1$ and for all strings $x_1, x_2, \ldots, x_{2k} \in \Sigma^*$ satisfying that for each j with $1 \leq j \leq 2k$, $x_{j+1} \in A$ implies $x_j \in A$, then B is hard for $\mathrm{P}_{||}^{\mathrm{NP}}$.

With the help of this lemma, the variants of many NP-complete optimization problems were proven to be $\mathrm{P}_{||}^{\mathrm{NP}}$-hard, for example the problem of determining whether the chromatic number of a graph G is odd [Wag87], or deciding the winner of Young elections [RSV03], and the winner of Carroll elections [HHR97a, HHR97b].

Theorem 3.3 *The problems* $\mathtt{DNP} - \mathtt{Odd}$, $\mathtt{DNP} - \mathtt{Equ}$, *and* $\mathtt{DNP} - \mathtt{Geq}$ *each are $\mathrm{P}_{||}^{\mathrm{NP}}$-complete.*

Proof. Each problem is easily seen to be a member of $\mathrm{P}_{||}^{\mathrm{NP}}$. Given as an oracle the NP set k-DNP, the domatic number of any graph G with n vertices can be determined exactly by n parallel queries to this set. [1] We will now prove the $\mathrm{P}_{||}^{\mathrm{NP}}$-hardness of the three problems given above. For $\mathtt{DNP} - \mathtt{Odd}$, this follows immediately from the reduction in the proof of Theorem 3.2, as the resulting graph H is odd if and only if an odd number of input graphs G_i with $1 \leq i \leq 2k$ is three-colorable. Applying Lemma 3.6, this shows that the set $\mathtt{DNP} - \mathtt{Odd}$ is $\mathrm{P}_{||}^{\mathrm{NP}}$-hard.

We now show that $\mathtt{DNP} - \mathtt{Equ}$ is $\mathrm{P}_{||}^{\mathrm{NP}}$-hard. This result immediately yields $\mathrm{P}_{||}^{\mathrm{NP}}$-hardness of the set $\mathtt{DNP} - \mathtt{Geq}$. Apply Lemma 3.6 with A being the NP-complete problem 3-COLOR and B being the set $\mathtt{DNP} - \mathtt{Equ}$. Fix $k \geq 1$

[1] As $\mathrm{P}_{||}^{\mathrm{NP}} = \mathrm{P}^{\mathrm{NP}[\mathcal{O}(\log)]}$, the corresponding algorithm using a DTM with the oracle NP set k-DNP can determine the domatic number of any graph by binary search in polynomial time.

and $2k$ graphs $G_1, G_2, \ldots, G_{2k}$ satisfying that if G_{i+1} is three-colorable, so is G_i, where $1 \leq i < 2k$. We can rewrite Equation (3.6) as

$$||\{i \mid x_i \in A\}|| \text{ is even} \quad \Leftrightarrow \quad f(x_1, x_2, \ldots, x_{2k}) \in B, \tag{3.7}$$

since the class $\mathrm{P}_{||}^{\mathrm{NP}}$ with its basis polynomial-time DTM is closed under complement.

We will construct a polynomial-time many-one reduction f satisfying Equation (3.7) as follows. Recall the construction from Theorem 3.2, where the first step was to apply reduction g from Lemma 3.1 on the input graphs G_i with $1 \leq i \leq 2k$ to generate graphs $H_i = g(G_i)$ satisfying the implications (3.1) and (3.2). Let $\times$ denote the associative operation on graphs that was defined in detail in the proof of Theorem 3.2 and was used to sum up the domatic numbers of the given graphs. Note that the triangle structure resulting from reduction g could be utilized to obtain such an operation. We define two graphs:

$$\begin{aligned} G_{\mathtt{odd}} &= H_1 \times H_3 \times \cdots \times H_{2k-1}, \\ G_{\mathtt{even}} &= H_2 \times H_4 \times \cdots \times H_{2k}. \end{aligned}$$

It remains to prove both directions from Equation (3.7). From left to right we have

$$\begin{aligned} &||\{i \mid G_i \in \text{3-}\mathtt{COLOR}\}|| \text{ is even} \\ &\quad \Longrightarrow \quad (\forall i : 1 \leq i \leq k)\ [\delta(H_{2i-1}) = \delta(H_{2i})] \\ &\quad \Longrightarrow \quad \sum_{1 \leq i \leq k} \delta(H_{2i-1}) = \sum_{1 \leq i \leq k} \delta(H_{2i}) \\ &\quad \Longrightarrow \quad \delta(G_{\mathtt{odd}}) = \delta(G_{\mathtt{even}}) \\ &\quad \Longrightarrow \quad \langle G_{\mathtt{odd}}, G_{\mathtt{even}} \rangle = f(G_1, G_2, \ldots, G_{2k}) \in \mathtt{DNP} - \mathtt{Equ}. \end{aligned}$$

From right to left we have:

$$\begin{aligned} &||\{i \mid G_i \in \text{3-}\mathtt{COLOR}\}|| \text{ is odd} \\ &\quad \Longrightarrow \quad (\exists i : 1 \leq i \leq k) \left[\begin{array}{l} \delta(H_{2i-1}) = 3 \wedge \delta(H_{2i}) = 2 \text{ and} \\ \delta(H_{2j-1}) = \delta(H_{2j}) \text{ for } j \neq i \end{array} \right] \\ &\quad \Longrightarrow \quad -1 + \sum_{1 \leq i \leq k} \delta(H_{2i-1}) = \sum_{1 \leq i \leq k} \delta(H_{2i}) \\ &\quad \Longrightarrow \quad \delta(G_{\mathtt{odd}}) - 1 = \delta(G_{\mathtt{even}}) \\ &\quad \Longrightarrow \quad \langle G_{\mathtt{odd}}, G_{\mathtt{even}} \rangle = f(G_1, G_2, \ldots, G_{2k}) \notin \mathtt{DNP} - \mathtt{Equ}. \end{aligned}$$

Lemma 3.6 implies that $\texttt{DNP} - \texttt{Equ}$ is $\mathrm{P}_{||}^{\mathrm{NP}}$-complete.

The same reduction also shows that $\texttt{DNP} - \texttt{Geq}$ is $\mathrm{P}_{||}^{\mathrm{NP}}$-hard, thus the set $\texttt{DNP} - \texttt{Geq}$ is $\mathrm{P}_{||}^{\mathrm{NP}}$-complete.

∎

Chapter 4

Generalized Dominating Sets

4.1 A General Framework

Graph partitioning problems appear in various forms and variants. To capture a large number of partitioning problems with one simple definition (in our particular case partitioning into dominating sets), Heggernes and Telle introduced a general notion in their paper "Partioning Graphs into Generalized Dominating Sets" [HT98]. Each set of the partition is characterized by two parameters σ and ρ, which restrict the number of vertices in the neighborhood of each vertex. As the problems to be defined are optimization problems (either maximum or mimimum), we are able to define exact versions of such generalized dominating sets. The complexity of the exact versions will be the topic of the next section.

Adopting the definitions by Heggernes and Telle [HT98], we now define the notions of (σ, ρ)-sets and (k, σ, ρ)-partitions.

Definition 4.1 *Let $G = (V, E)$ be a given graph, let $\sigma \subseteq \mathbb{N}$ and $\rho \subseteq \mathbb{N}$ be given sets, and let k be a positive integer.*

- *A subset $V' \subseteq V$ of the vertices of G is said to be a* (σ, ρ)-set *if and only if for each $v \in V'$, $||N(v) \cap V'|| \in \sigma$, and for each $v \notin V'$, it is $||N(v) \cap V'|| \in \rho$.*

- *A* (k, σ, ρ)-partition of *G is a partition of V into k pairwise disjoint subsets $V_1, V_2, \ldots, V_k$ with $V = \cup_{i=1}^{k} V_i$ such that for each i the set V_i is a (σ, ρ)-set, where $1 \leq i \leq k$.*

- *Define the problem*

$$(k, \sigma, \rho)\text{-}\texttt{Partition} = \{G \mid G \textit{ is a graph that has a } (k, \sigma, \rho)\textit{-partition}\}.$$

The values of particular interest for the parameters σ and ρ are the sets $\{0\}$, $\{0,1\}$, $\{1\}$, $\mathbb{N}$, and $\mathbb{N}^+$, however the value $\rho = \{0\}$ is not considered. Note that for each σ, problem $(k, \sigma, \{0\})$-`Partition` is restricted to graphs $G = (V, E)$ such that the vertex set V can be partitioned into k disjoint cliques $V_1, V_2, \ldots, V_k$ of size $i + 1$, where $i \in \sigma$. This property can be recognized in polynomial time for any value of k and σ, thus these problems are easy to solve and need not to be examined any further.

4.2 NP-Completeness Results

In Definition 4.1, restrict parameter σ to the sets $\{0\}$, $\{0,1\}$, $\{1\}$, $\mathbb{N}$, and $\mathbb{N}^+$, and restrict parameter ρ to the sets $\{0,1\}$, $\{1\}$, $\mathbb{N}$, and $\mathbb{N}^+$. For many of the $5 \cdot 4 = 20$ possible combinations, Heggernes and Telle provide the exact bound where the corresponding problem (k, σ, ρ)-`Partition` switches from being efficiently solvable to intractability [HT98]. That is, they determine the precise value k for which the set (k, σ, ρ)-`Partition` is complete for NP, but the set $(k-1, \sigma, \rho)$-`Partition` is solvable within polynomial time. Table 4.1 gives an overview of their (and previously known) results.

σ \ ρ	$\mathbb{N}$	$\mathbb{N}^+$	$\{1\}$	$\{0,1\}$
$\mathbb{N}$	∞^-	3^+	2	∞^-
$\mathbb{N}^+$	∞^-	2^+	2	∞^-
$\{1\}$	2^-	2	3	3^-
$\{0,1\}$	2^-	2	3	3^-
$\{0\}$	3^-	3	4	4^-

Table 4.1: NP-completeness for the problems (k, σ, ρ)-`Partition`

Recall Definition 2.13 of the domatic number problem, which is a partitioning problem as well. For a given graph $G = (V, E)$, each dominating set D of G has to satisfy the condition that every vertex $v \in V - D$ has to be adjacent to at least one vertex in D. That is, $||N[v] \cap D|| \geq 1$ which can

be rephrased to $||N(v) \cap D|| \in \mathbb{N}^+$, as it is $v \in V - D$. Since no constraint exists regarding the neighborhood of vertices $v \in D$, k-DNP is nothing else than problem $(k, \mathbb{N}, \mathbb{N}^+)$-Partition. As we know from Lemma 3.1, the domatic number problem k-DNP is NP-complete for all $k \geq 3$, whereas Fact 2.1 states that 2-DNP lies in P. Therefore, the corresponding entry in Table 4.1 is "3" for $\sigma = \mathbb{N}$ and $\rho = \mathbb{N}^+$.

A value of ∞ in Table 4.1 means that this problem is efficiently solvable for all values of k. The marks "+" and "−" on some problems can be explained after the following definition.

Definition 4.2 *Let k be a positive integer and σ and ρ be sets chosen among $\{0\}$, $\{1\}$, $\{0,1\}$, $\mathbb{N}$, and $\mathbb{N}^+$. We say that (k, σ, ρ)-Partition is a* minimum problem *if and only if*

$$(k, \sigma, \rho)\text{-Partition} \subseteq (k+1, \sigma, \rho)\text{-Partition}$$

for each $k \geq 1$, and we say that (k, σ, ρ)-Partition is a maximum problem *if and only if*

$$(k+1, \sigma, \rho)\text{-Partition} \subseteq (k, \sigma, \rho)\text{-Partition}$$

for each $k \geq 1$.

The problems in Table 4.1 that are marked by a "+" are maximum problems, and the problems that are marked by a "−" are minimum problems in the above sense. The following fact states this property, and it will be verified by the subsequent short proof.

Fact 4.1 *The following facts are true for the set (k, σ, ρ)-Partition:*

- *For each positive integer k, for each $\sigma \in \{\mathbb{N}, \mathbb{N}^+, \{0\}, \{0,1\}, \{1\}\}$, and for each $\rho \in \{\mathbb{N}, \{0,1\}\}$, it holds that*

$$(k, \sigma, \rho)\text{-Partition} \subseteq (k+1, \sigma, \rho)\text{-Partition}.$$

- *For each positive integer k and for each $\sigma \in \{\mathbb{N}, \mathbb{N}^+\}$, it holds that*

$$(k+1, \sigma, \mathbb{N}^+)\text{-Partition} \subseteq (k, \sigma, \mathbb{N}^+)\text{-Partition}.$$

Proof. To see that all sets (k, σ, ρ)-Partition with $\rho = \mathbb{N}$ are minimum problems, note that we obtain a $(k+1, \sigma, \mathbb{N})$-partition from a $(k, \sigma, \mathbb{N})$-partition by simply adding the empty set $V_{k+1} = \emptyset$. The proof for the case $\rho = \{0, 1\}$ is analogous.

To prove that the (k, σ, ρ)-Partition problems with $\sigma \in \{\mathbb{N}, \mathbb{N}^+\}$ and $\rho = \mathbb{N}^+$ are maximum problems, note that once we have found a partition into $k+1$ pairwise disjoint $(\sigma, \mathbb{N}^+)$-sets $V_1, V_2, \ldots, V_{k+1}$, the sets $V_1, V_2, \ldots, V_{k-1}, V'_k$ with $V'_k = V_k \cup V_{k+1}$ remain $(\sigma, \mathbb{N}^+)$-sets as well and thus form a $(k, \sigma, \mathbb{N}^+)$-partition.

∎

The entries in Table 4.1 that are marked neither by a "+" nor by a "−" are neither maximum nor minimum problems in the sense defined above. That is, it does neither hold that

$$(k+1, \sigma, \rho)\text{-Partition} \subseteq (k, \sigma, \rho)\text{-Partition},$$

nor does it hold that

$$(k, \sigma, \rho)\text{-Partition} \subseteq (k+1, \sigma, \rho)\text{-Partition},$$

since for each $k \geq 1$, there exist graphs G such that G is a member of the set (k, σ, ρ)-Partition but G is not contained in (ℓ, σ, ρ)-Partition for any $\ell \geq 1$ with $\ell \neq k$. We will give specific examples for graphs with these properties in three of the cases which are unmarked in Table 4.1.

For example, consider the set $(k, \{1\}, \{1\})$-Partition. By definition, this problem contains all graphs $G = (V, E)$ that can be partitioned into k subsets $V_1, V_2, \ldots, V_k$ such that, for each i, if $v \in V_i$ then $||N(v) \cap V_i|| = 1$, and if $v \notin V_i$ then $||N(v) \cap V_i|| = 1$. It follows that every graph in the set $(k, \{1\}, \{1\})$-Partition must be k-regular; that is, every vertex has degree k, see Definition 2.7. Hence, for all $k \geq 1$, $(k, \{1\}, \{1\})$-Partition and $(k+1, \{1\}, \{1\})$-Partition are disjoint, so neither

$$\begin{aligned} (k, \{1\}, \{1\})\text{-Partition} &\subseteq (k+1, \{1\}, \{1\})\text{-Partition}, \text{ nor} \\ (k+1, \{1\}, \{1\})\text{-Partition} &\subseteq (k, \{1\}, \{1\})\text{-Partition}. \end{aligned}$$

The complete graph K_n with n vertices connects each pair of vertices with an edge, that is, every pair of vertices is adjacent. Looking at problem $(k, \{0\}, \mathbb{N}^+)$-Partition, graph K_n is in $(n, \{0\}, \mathbb{N}^+)$-Partition but not in $(k, \{0\}, \mathbb{N}^+)$-Partition for any $k \geq 1$ with $k \neq n$. Almost the same argument applies to the case $\sigma = \mathbb{N}$ and $\rho = \{1\}$, except that

now K_n is contained in $(k, \mathbb{N}, \{1\})$-Partition for $k \in \{1, n\}$ but not in $(\ell, \mathbb{N}, \{1\})$-Partition for any $\ell \geq 1$ with $\ell \notin \{1, n\}$. Similar arguments work in the other cases.

As we are interested in the exact versions of the generalized dominating set problems, we will focus our attention on (k, σ, ρ)-Partition problems that are minimum or maximum problems in the above sense. The challenge for each graph G is to find the exact cutoff point k such that it holds $G \in (k, \sigma, \rho)$-Partition but $G \notin (k+1, \sigma, \rho)$-Partition in the maximum case, and $G \in (k, \sigma, \rho)$-Partition but $G \notin (k-1, \sigma, \rho)$-Partition in the minimum case respectively.

4.3 Exact Partitioning Problems

First we will formally define the exact versions of the generalized dominating set problems.

Definition 4.3 *Let σ and ρ be sets chosen among $\mathbb{N}$, $\mathbb{N}^+$, $\{0\}$, $\{0,1\}$, and $\{1\}$, and let k be a positive integer. We define the* exact version of the problem (k, σ, ρ)-Partition *by*

$$\begin{aligned}&\text{Exact-}(k, \sigma, \rho)\text{-Partition}\\ &\quad = (k, \sigma, \rho)\text{-Partition} \cap \overline{(k-1, \sigma, \rho)\text{-Partition}}\end{aligned}$$

if $k \geq 2$ and (k, σ, ρ)-Partition is a minimum problem and

$$\begin{aligned}&\text{Exact-}(k, \sigma, \rho)\text{-Partition}\\ &\quad = (k, \sigma, \rho)\text{-Partition} \cap \overline{(k+1, \sigma, \rho)\text{-Partition}}\end{aligned}$$

if $k \geq 1$ and (k, σ, ρ)-Partition is a maximum problem.

For instance, $(k, \mathbb{N}, \mathbb{N}^+)$-Partition is equal to k-DNP, which is a maximization problem. Its exact version Exact-t-DNP was dealt with in the previous section. In contrast, the problem $(k, \{0\}, \mathbb{N})$-Partition is equal to the minimization k-colorability problem, where graph G needs to be partitioned into k vertex disjoint independent sets. The computational complexity of the problem Exact-$(k, \{0\}, \mathbb{N})$-Partition was determined precisely by Rothe [Rot03], who proved the DP-completeness for all values $k \geq 4$. Since the problem Exact-$(k, \{0\}, \mathbb{N})$-Partition lies within NP for $k \leq 3$, the exact version of the colorability problem cannot be DP-complete for $k \leq 3$ unless the boolean hierarchy over NP collapses.

The following fact points out DP membership for all problems which can be defined in the exact sense as in Definition 4.3. To prove completeness for the class DP, it then remains to show that each of these problems is DP-hard, which will be established for three more sets in this section.

Fact 4.2 *Let σ and ρ be sets chosen among $\mathbb{N}$, $\mathbb{N}^+$, $\{0\}$, $\{0,1\}$, and $\{1\}$, let k be a positive integer, and let (k,σ,ρ)-*`Partition` *be a maximum or minimum problem. Then,* `Exact`*-(k,σ,ρ)-*`Partition` *is contained in* DP*.*

4.3.1 The Case $\sigma = \mathbb{N}^+$ and $\rho = \mathbb{N}^+$

In contrast to the domatic number problem, every set of a partition of a given graph G into $(\mathbb{N}^+,\mathbb{N}^+)$-sets $V_1, V_2, \ldots, V_k$ must not only be a dominating set of G, but V_i additionally has to dominate itself in the sense that each vertex $u \in V_i$ has to be adjacent to another vertex $v \in V_i$. Clearly the set $(k,\mathbb{N}^+,\mathbb{N}^+)$-`Partition` is a maximum problem, see Fact 4.1. Analogously to the definition of the domatic number $\delta(G)$ of a graph G, see Definition 2.13, we will introduce a parameter for the maximum number of $(\mathbb{N}^+,\mathbb{N}^+)$-sets into which a given graph G can be partitioned.

Definition 4.4 *For every graph G, we define the maximum value k for which G has a $(k,\mathbb{N}^+,\mathbb{N}^+)$-partition as follows:*

$$\gamma(G) = \max\{k \in \mathbb{N}^+ \mid G \in (k,\mathbb{N}^+,\mathbb{N}^+)\text{-}\texttt{Partition}\}.$$

Theorem 4.1 *The problem* `Exact`*-$(k,\mathbb{N}^+,\mathbb{N}^+)$-*`Partition` *is* DP*-complete for each $k \geq 3$.*

Proof. By Fact 4.2, the set `Exact`-$(k,\mathbb{N}^+,\mathbb{N}^+)$-`Partition` is a member of the class DP, therefore it leaves to show that `Exact`-$(k,\mathbb{N}^+,\mathbb{N}^+)$-`Partition` is DP-hard for $k \geq 3$.

First we show that `Exact`-$(k,\mathbb{N}^+,\mathbb{N}^+)$-`Partition` can easily be reduced to `Exact`-$(k+1,\mathbb{N}^+,\mathbb{N}^+)$-`Partition`, thus it suffices to prove the theorem for the case $k = 3$. The following polynomial-time many-one reduction g maps an input graph G to $G' = g(G)$ such that $\gamma(G') = \gamma(G) + 1$. Given $G = (V,E)$, define two vertex disjoint graphs $G_1 = G \oplus x = (V_1,E_1)$ and $G_2 = G \oplus y = (V_2,E_2)$, where x and y are additional vertices that are connected to every other vertex representing graph G. We conclude reduction g by setting $G' = (V',E')$ with

$$\begin{aligned} V' &= V_1 \cup V_2, \\ E' &= E_1 \cup E_2 \cup \{x,y\}. \end{aligned}$$

It is easy to see that $\gamma(G) = k$ implies $\gamma(G') \geq \gamma(G)+1 = k+1$. Given the $(\mathbb{N}^+, \mathbb{N}^+)$-sets $D_1, D_2, \ldots, D_k$ for graph G, keep all vertices in V' originating from graph G in the same sets D_i, $1 \leq i \leq k$, which are all vertices in V_1 except x, and all vertices in V_2 except y, respectively. Then define the set $D_{k+1} = \{x, y\}$, which is a $(\mathbb{N}^+, \mathbb{N}^+)$-set for G' as well. To show that it holds $\gamma(G') \leq \gamma(G) + 1$, suppose that $\gamma(G') = \gamma(G) + 2 = k + 2$ by the $(\mathbb{N}^+, \mathbb{N}^+)$-sets $D_1, D_2, \ldots, D_{k+2}$. Let $x \in D_i$ for $i \in \{1, 2, \ldots k+2\}$. Looking at the subgraph $G'[V_1 - \{x\}] = G$, put all vertices belonging to the set D_i to another $(\mathbb{N}^+, \mathbb{N}^+)$-set D_j with $j \in \{1, 2, \ldots k+2\}$ and $j \neq i$. This yields $k+1$ vertex disjoint sets of G which are $(\mathbb{N}^+, \mathbb{N}^+)$-sets. This is a contradiction to the assumption $\gamma(G) = k$.

Having established this easy chain of reductions, we now return to the main proof and show that `Exact-`$(k, \mathbb{N}^+, \mathbb{N}^+)$`-Partition` is DP-hard for the value $k = 3$. In their paper [HT98], Heggernes and Telle presented a reduction from the problem `NAE-3-SAT` to the problem $(2, \mathbb{N}^+, \mathbb{N}^+)$`-Partition` to prove the latter problem NP-complete, see Definition 2.9 for the description of the NP-complete problem `NAE-3-SAT`. We will use a slight modification of this reduction to prove the claim of Theorem 4.1. As input, let two boolean formulas $\varphi_1 = (X, \hat{C})$ and $\varphi_2 = (Y, \hat{D})$ be given with disjoint variable sets $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_1, y_2, \ldots, y_r\}$. The clause sets are defined by $\hat{C} = \{c_1, c_2, \ldots, c_m\}$ and $\hat{D} = \{d_1, d_2, \ldots, d_s\}$. Without loss of generality, we assume that each of the two variable sets has cardinality at least two, and that every literal appears in at least one clause. Otherwise we can either add additional variables to the set, or in the second case, for each literal l_i not appearing in any clause, we can add $(l_i \vee \overline{l_i})$ to the clause set without altering membership in `NAE-3-SAT` for φ_1 or φ_2.

For any clause $c = (l_1 \vee l_2 \vee l_3)$, define $\check{c} = (\overline{l_1} \vee \overline{l_2} \vee \overline{l_3})$, where $\overline{l_1}$, $\overline{l_2}$, and $\overline{l_3}$, respectively, denotes the negation of the literal l_1, l_2, and l_3. Note that for the same truth assignment t, clause c is satisfied in the not-all-equal sense under t if and only if assignment t satisfies $\check{c}$ in the same sense. Therefore, when we define the clause sets $\check{C} = \{\check{c}_1, \check{c}_2, \ldots, \check{c}_m\}$ and $\check{D} = \{\check{d}_1, \check{d}_2, \ldots, \check{d}_s\}$, as well as $C = \hat{C} \cup \check{C}$ and $D = \hat{D} \cup \check{D}$, we can conclude that:

$$\begin{aligned}(X, C) \in \texttt{NAE-3-SAT} &\iff (X, \hat{C}) \in \texttt{NAE-3-SAT} \\ &\iff (X, \check{C}) \in \texttt{NAE-3-SAT}\end{aligned}$$

and

$$\begin{aligned}(Y, D) \in \texttt{NAE-3-SAT} &\iff (Y, \hat{D}) \in \texttt{NAE-3-SAT} \\ &\iff (Y, \check{D}) \in \texttt{NAE-3-SAT}.\end{aligned}$$

Again we apply Lemma 3.2 with $k = 1$ being fixed, with NAE-3-SAT being the NP-complete problem A, and with Exact-$(3, \mathbb{N}^+, \mathbb{N}^+)$-Partition being the set B from this lemma. Let two boolean formulas φ_1 and φ_2 be given such that $\varphi_2 \in$ NAE-3-SAT implies $\varphi_1 \in$ NAE-3-SAT. We will construct a polynomial-time many-one reduction f which maps the two boolean formulas φ_1 and φ_2 to a graph $G = f(\varphi_1, \varphi_2)$ with the property:

$$(\varphi_1 \in \text{NAE-3-SAT} \wedge \varphi_2 \notin \text{NAE-3-SAT}) \iff \gamma(G) = 3. \tag{4.1}$$

We will describe reduction f in the following. At first we will extend the two formulas $\varphi_1 = (X, \hat{C})$ and $\varphi_2 = (Y, \hat{D})$ as in the construction given above by introducing the "negated" clauses and hereby doubling the clause sets $\hat{C}$ and $\hat{D}$ by adding $\check{C}$ and $\check{D}$. We get $\varphi_1' = (X, \hat{C} \cup \check{C})$ and $\varphi_2' = (Y, \hat{D} \cup \check{D})$. Next, we create an 8-clique A with vertices $a_1, a_2, \ldots, a_8$ for φ_1' and an eight clique B with vertices $b_1, b_2, \ldots, b_8$ for φ_2'. Let the variable sets of the formulas be $X = \{x_1, x_2, \ldots, x_n\}$, and $Y = \{y_1, y_2, \ldots, y_r\}$ respectively. The clause sets $C = \hat{C} \cup \check{C}$ and $D = \hat{D} \cup \check{D}$ consist of $\hat{c}_1, \hat{c}_2, \ldots, \hat{c}_m, \check{c}_1, \check{c}_2, \ldots, \check{c}_m$, and $\hat{d}_1, \hat{d}_2, \ldots, \hat{d}_s, \check{d}_1, \check{d}_2, \ldots, \check{d}_s$ respectively. For each i with $1 \leq i \leq n$, two vertices x_i and $\overline{x}_i$ are created for the variable x_i. For each j, $1 \leq j \leq r$, two vertices, y_j and $\overline{y}_j$ are created for the variable y_j. We connect x_i and $\overline{x}_i$ to the two vertices a_1 and a_2 of A, also each of the vertices y_j and $\overline{y}_j$ is connected to the two vertices b_1 and b_2 of B. For each pair of variables $\{x_i, y_j\}$, we create one vertex $u_{i,j}$ that is connected to the four vertices x_i, $\overline{x}_i$, y_j, and $\overline{y}_j$, thereby adding $n \cdot r$ more vertices to the construction. Finally, we create one vertex for each clause in C and D, thus the vertices $\hat{c}_i$ and $\check{c}$ with $1 \leq i \leq m$, and $\hat{d}_j$ and $\check{d}_j$ with $1 \leq j \leq s$ are created for each clause in C and D. We connect each of these clause vertices to the vertices representing the literals it contains. Analogously to the literal vertices, each of the clause vertices $\hat{c}_i$ and $\check{c}_i$ with $1 \leq i \leq m$ is connected to a_1 and a_2, and each of the clause vertices $\hat{d}_j$ and $\check{d}_j$, $1 \leq j \leq s$, is connected to b_1 and b_2. This completes the construction of the graph $G = f(\varphi_1, \varphi_2)$.

Figure 4.1, which is taken from [RR06b], shows the graph G resulting from the reduction f applied to the two formulas

$$\begin{aligned} \varphi_1 &= (x_1 \vee \overline{x}_2 \vee x_3) \wedge (\overline{x}_1 \vee x_2 \vee x_3) \quad \text{and} \\ \varphi_2 &= (y_1 \vee y_2 \vee y_3) \wedge (\overline{y}_1 \vee \overline{y}_2 \vee \overline{y}_3). \end{aligned}$$

Note that $\gamma(G) \leq 4$, since the degree of each $u_{i,j}$ is four. We have three cases to distinguish, as was the case in the proof for the DP-completeness of the exact domatic number problem in Theorem 3.1.

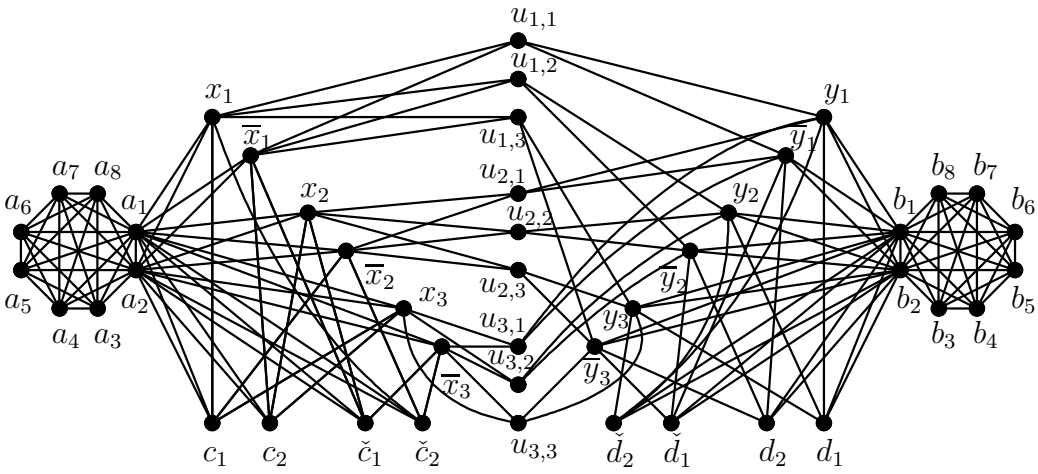

Figure 4.1: Graph $G = f(\varphi_1, \varphi_2)$

Case 1: [$\boldsymbol{\varphi_1} \in$ NAE-3-SAT **and** $\boldsymbol{\varphi_2} \in$ NAE-3-SAT] Since $\gamma(G) \leq 4$ as noted above, it suffices to construct four $(\mathbb{N}^+, \mathbb{N}^+)$-sets for G to prove that $\gamma(G) = 4$. Let t be a truth assignment for φ_1, and let t' be a truth assignment for φ_2, satisfying the corresponding formulas in the not-all-equal sense. Our partition into four $(\mathbb{N}^+, \mathbb{N}^+)$-sets V_1, V_2, V_3, and V_4 goes as follows:

$$\begin{aligned}
V_1 &= \hat{C} \cup \check{C} \cup \{a_5, a_6\} \cup \{b_1, b_3\} \\
&\quad \cup \{x \mid x \text{ is a literal over } X \text{ and } t(x) = \text{true}\}, \\
V_2 &= \{u_{i,j} \mid (1 \leq i \leq n-1 \wedge j = 1) \vee (i = n \wedge 2 \leq j \leq r)\} \\
&\quad \cup \{a_7, a_8\} \cup \{b_2, b_4\} \\
&\quad \cup \{x \mid x \text{ is a literal over } X \text{ and } t(x) = \text{false}\}, \\
V_3 &= \hat{D} \cup \check{D} \cup \{a_1, a_3\} \cup \{b_5, b_6\} \\
&\quad \cup \{y \mid y \text{ is a literal over } Y \text{ and } \tilde{t}(y) = \text{true}\}, \\
V_4 &= \{u_{i,j} \mid (i = n \wedge j = r) \vee (1 \leq i \leq n-1 \wedge 2 \leq j \leq r)\} \\
&\quad \cup \{a_2, a_4\} \cup \{b_7, b_8\} \\
&\quad \cup \{y \mid y \text{ is a literal over } Y \text{ and } \tilde{t}(y) = \text{false}\}.
\end{aligned}$$

Thus, $\gamma(G) \geq 4$. As noted above, this implies $\gamma(G) = 4$ in this case.

Case 2: [$\boldsymbol{\varphi_1} \in$ NAE-3-SAT **and** $\boldsymbol{\varphi_2} \notin$ NAE-3-SAT] Let t be a truth assignment satisfying φ_1. Just as in Case 1, we can partition G into three

$(\mathbb{N}^+, \mathbb{N}^+)$-sets V_1, V_2, and V_3 as follows to prove $\gamma(G) \geq 3$:

$$\begin{aligned} V_1 &= \hat{C} \cup \check{C} \cup \{a_5, a_6\} \cup \{b_1, b_3\} \\ &\quad \cup \{x \mid x \text{ is a literal over } X \text{ and } t(x) = \text{true}\}, \\ V_2 &= \{u_{i,j} \mid 1 \leq i \leq n \wedge 1 \leq j \leq r\} \cup \{a_7, a_8\} \cup \{b_2, b_4\} \\ &\quad \cup \{x \mid x \text{ is a literal over } X \text{ and } t(x) = \text{false}\}, \\ V_3 &= \hat{D} \cup \check{D} \cup \{a_1, a_2, a_3, a_4\} \cup \{b_5, b_6, b_7, b_8\} \\ &\quad \cup \{y \mid y \text{ is a literal over } Y\}. \end{aligned}$$

Thus, $3 \leq \gamma(G) \leq 4$. To show that $\gamma(G) = 3$, suppose that it holds $\gamma(G) = 4$, and let the partition of G into four $(\mathbb{N}^+, \mathbb{N}^+)$-sets be given by U_1, U_2, U_3, and U_4. Vertex $u_{1,1}$ is adjacent to exactly four vertices, namely to x_1, $\overline{x}_1$, y_1, and $\overline{y}_1$. These four vertices must then be in four distinct sets of the partition. Without loss of generality, suppose that $x_1 \in U_1$, $\overline{x}_1 \in U_2$, $y_1 \in U_3$, and $\overline{y}_1 \in U_4$. For each j with $2 \leq j \leq r$, the vertices y_j and $\overline{y}_j$ are connected to x_1 and $\overline{x}_1$ via vertex $u_{1,j}$, so it follows that either $y_j \in U_3$ and $\overline{y}_j \in U_4$, or $y_j \in U_4$ and $\overline{y}_j \in U_3$.

Every clause vertex $\hat{d}_j$, $1 \leq j \leq r$, is connected only to the vertices representing its literals and to the vertices b_1 and b_2, which therefore must be in the sets U_1 and U_2, respectively. Thus, every clause vertex $\hat{d}_j$ is connected to at least one literal vertex in U_3 and to at least one literal vertex in U_4. This describes a valid truth assignment for φ_2 in the not-all-equal sense. This is a contradiction to the case assumption that $\varphi_2 \notin$ NAE-3-SAT.

Case 3: [$\boldsymbol{\varphi_1 \notin}$ NAE-3-SAT **and** $\boldsymbol{\varphi_2 \notin}$ NAE-3-SAT] Analogously to the above two cases, we give a partition of G into two $(\mathbb{N}^+, \mathbb{N}^+)$-sets to show that $\gamma(G) \geq 2$:

$$\begin{aligned} V_1 &= \{u_{i,j} \mid 1 \leq i \leq n \wedge 1 \leq j \leq r\} \cup \{x_i \mid 1 \leq i \leq n\} \\ &\quad \cup \{y_j \mid 1 \leq j \leq r\} \cup \{a_1, a_3, a_5, a_7\} \cup \{b_1, b_3, b_5, b_7\}, \\ V_2 &= \hat{C} \cup \check{C} \cup \hat{D} \cup \check{D} \cup \{\overline{x}_i \mid 1 \leq i \leq n\} \cup \{\overline{y}_j \mid 1 \leq j \leq r\} \\ &\quad \cup \{a_2, a_4, a_6, a_7\} \cup \{b_2, b_4, b_6, b_8\}. \end{aligned}$$

It follows that $2 \leq \gamma(G) \leq 4$. By the same argument as in the second case, we can show that $\gamma(G) \neq 4$. To prove that it even holds that $\gamma(G) < 3$, suppose that $\gamma(G) = 3$, and let the partition of G

into three $(\mathbb{N}^+, \mathbb{N}^+)$- sets be given by U_1, U_2, and U_3. Without loss of generality, assume that the vertices x_1 and $\overline{x}_1$ belong to distinct sets U_i. Otherwise, following the argumentation in the second case, each y_j and $\overline{y}_j$ must belong to distinct sets U_a and U_b, with $a \neq b$, since $u_{1,j}$ is only adjacent to the vertices x_1, $\overline{x}_1$, y_j, and $\overline{y}_j$. This would lead to a symmetric argument, thus suppose it is $x_1 \in U_1$ and $\overline{x}_1 \in U_2$.

It follows that for each j with $1 \leq j \leq r$, at least one of y_j or $\overline{y}_j$ has to be in U_3. We will pick one subcase and prove that it cannot hold that $\gamma(G) = 3$. All other subcases can be treated analogously. We will restrict our attention to the case that both vertices y_j and $\overline{y}_j$ are in U_3. It follows that

$$(\forall i : 1 \leq i \leq n)\,[(x_i \in U_1 \wedge \overline{x}_i \in U_2) \vee (x_i \in U_2 \wedge \overline{x}_i \in U_1)]. \quad (4.2)$$

Since it is $\varphi_1 \notin$ NAE-3-SAT, there exists at least one clause $\hat{c}_i \in \hat{C}$ for each truth assignment t for φ_1 such that $\hat{c}_i = (l_1 \vee l_2 \vee l_3)$ and the literals l_1, l_2, and l_3 are either simultaneously true or simultaneously false under t. The same is true for the corresponding negated clause $\check{c}_i$, since it contains the negations of the literals l_1, l_2, and l_3, and all three truth values are flipped under t. In other words, it is $t(\overline{l_1}) = 1 - t(l_1)$, $t(\overline{l_2}) = 1 - t(l_2)$, and $t(\overline{l_3}) = 1 - t(l_3)$. Since the corresponding clause vertex $\hat{c}_i$ is adjacent to l_1, l_2, l_3, a_1, and a_2, it follows that l_1, l_2, and l_3 are in the same set of the partition, say in U_1. Thus it follows that either $a_1 \in U_2$ and $a_2 \in U_3$, or $a_1 \in U_3$ and $a_2 \in U_2$. Similarly, since the clause vertex $\check{c}_i$ is adjacent to $\overline{l_1}$, $\overline{l_2}$, $\overline{l_3}$, a_1, and a_2, the vertices $\overline{l_1}$, $\overline{l_2}$, $\overline{l_3}$ are in the same set of the partition that must be distinct from U_1. Let U_2, say, be this set. It follows that either $a_1 \in U_1$ and $a_2 \in U_3$, or $a_1 \in U_3$ and $a_2 \in U_1$, which is a contradiction.

Each of the remaining subcases can be reduced to the case in Equation (4.2), and the above contradiction follows. Hence we can conclude $\gamma(G) = 2$.

By construction the case "$\varphi_1 \notin$ NAE-3-SAT and $\varphi_2 \in$ NAE-3-SAT" cannot occur, since this would contradict that $\varphi_2 \in$ NAE-3-SAT implies $\varphi_1 \in$ NAE-3-SAT. Thus, the case distinction is complete and we obtain:

$$\begin{aligned} &||\{i \mid \varphi_i \in \text{NAE-3-SAT}\}|| \text{ is odd} \\ &\iff \varphi_1 \in \text{NAE-3-SAT} \wedge \varphi_2 \notin \text{NAE-3-SAT} \\ &\iff \gamma(G) = 3, \end{aligned}$$

which proves the correctness of Equation (4.1) and thereby fulfilling Equation (3.3) of Lemma 3.2. This completes our proof of DP-completeness of the set Exact-$(3, \mathbb{N}^+, \mathbb{N}^+)$-Partition.

∎

As was the case with the exact domatic number problem, we can give a bound where Exact-$(k, \mathbb{N}^+, \mathbb{N}^+)$-Partition cannot be DP-complete unless the boolean hierarchy over NP collapses. Again, we fail to determine a precise cutoff point where the exact version of the maximum problem migrates from DP-completeness to coNP-membership. Still, we observe that the set Exact-$(1, \mathbb{N}^+, \mathbb{N}^+)$-Partition is in coNP (and even coNP-complete) and therefore cannot be DP-complete unless the boolean hierarchy over NP collapses to its first level.

Theorem 4.2 Exact-$(1, \mathbb{N}^+, \mathbb{N}^+)$-Partition *is* coNP*-complete.*

Proof. Exact-$(1, \mathbb{N}^+, \mathbb{N}^+)$-Partition is in coNP, as it can be written as the join of one set in P and one set in coNP

$$\text{Exact-}(1, \mathbb{N}^+, \mathbb{N}^+)\text{-Partition} = A \cap \overline{B},$$

where $A = (1, \mathbb{N}^+, \mathbb{N}^+)$-Partition is decidable in polynomial time and thus in P, and the set $B = (2, \mathbb{N}^+, \mathbb{N}^+)$-Partition is a member of NP by simply nondeterministically guessing a partition into two $(\mathbb{N}^+, \mathbb{N}^+)$-sets. Notice that the coNP-hardness of Exact-$(1, \mathbb{N}^+, \mathbb{N}^+)$-Partition is established by the original reduction from NAE-3-SAT to $(2, \mathbb{N}^+, \mathbb{N}^+)$-Partition presented by Heggernes and Telle [HT98].

∎

In the next subsections of this chapter, we will analyze the minimum problems Exact-$(k, \sigma, \mathbb{N})$-Partition, where σ is chosen among the sets $\{\mathbb{N}, \mathbb{N}^+, \{0\}, \{0, 1\}, \{1\}\}$. Depending on the value of $k \geq 2$, we ask how hard it is to decide whether a given graph G has a $(k, \sigma, \mathbb{N})$-partition but not a $(k-1, \sigma, \mathbb{N})$-partition.

4.3.2 The Case $\sigma = \{0\}$ and $\rho = \mathbb{N}$

The case $\sigma = \{0\}$ corresponds to the k-colorability problem. Given as input a graph $G = (V, E)$, we need to find a partition into the minimum number of independent sets. Strengthening the result obtained by Wagner [Wag87], Rothe obtained the following DP-completeness result for the exact version of the colorability problem [Rot03].

Theorem 4.3 *The problem* Exact-k-COLOR *is* DP*-complete for* $k \geq 4$.

Generalizing this result, Rothe [Rot03] further obtained completeness results in the boolean hierarchy over NP for the sets Exact-M_k-COLOR, where for a given graph G and k positive integers contained in the set M_k, the question is whether $\chi(G)$ is equal to one of the values in M_k.

Theorem 4.4 *For fixed* $k \geq 1$ *and* $M_k = \{3k+1, 3k+3, \ldots, 5k-1\}$*, the problem* Exact-$M_k$-COLOR *is* $\mathrm{BH}_{2k}(\mathrm{NP})$*-complete.*

4.3.3 The Cases $\sigma \in \{\mathbb{N}, \mathbb{N}^+\}$ and $\rho = \mathbb{N}$

As noted by Heggernes and Telle [HT98], the corresponding minimization problems $(k, \mathbb{N}, \mathbb{N})$-Partition and $(k, \mathbb{N}^+, \mathbb{N})$-Partition are solvable in polynomial time for each $k \geq 1$, which outright implies that the exact versions Exact-$(k, \mathbb{N}, \mathbb{N})$-Partition and Exact-$(k, \mathbb{N}^+, \mathbb{N})$-Partition lie in the class P as well.

4.3.4 The Case $\sigma = \{0, 1\}$ and $\rho = \mathbb{N}$

We will first define a parameter for every graph G that equals the minimum number of $(\{0,1\}, \mathbb{N})$-sets into which graph G can be partitioned.

Definition 4.5 *For every graph* G*, define the minimum value of* k *for which* G *has a* $(k, \{0,1\}, \mathbb{N})$*-partition as follows:*

$$\alpha(G) \;=\; \min\{k \in \mathbb{N}^+ \mid G \in (k, \{0,1\}, \mathbb{N})\text{-Partition}\}.$$

Theorem 4.5 *For each* $k \geq 5$*, the set* Exact-$(k, \{0,1\}, \mathbb{N})$-Partition *is complete for* DP.

Proof. By Fact 4.2, Exact-$(k, \{0,1\}, \mathbb{N})$-Partition is contained in DP for all $k \geq 1$. To prove that Exact-$(k, \{0,1\}, \mathbb{N})$-Partition is DP-hard for all $k \geq 5$, it suffices to prove DP-hardness for the case $k = 5$. A polynomial-time many-one reduction g from Exact-$(k, \{0,1\}, \mathbb{N})$-Partition to Exact-$(k+1, \{0,1\}, \mathbb{N})$-Partition is easily given by simply adding a vertex v to the input graph $G = (V, E)$ and connecting it to every other vertex in V. If we call the resulting graph $G' = g(G)$, it is $\alpha(G') = \alpha(G) + 1$. To prove DP-hardness of the set Exact-$(5, \{0,1\}, \mathbb{N})$-Partition, we again apply Wagner's Lemma 3.2 with $k = 1$ being fixed, with 1-3-SAT being

the NP-complete problem A, and with Exact-$(5, \{0,1\}, \mathbb{N})$-Partition being the set B from this lemma. The NP-complete problem 1-3-SAT was described in Definition 2.9 of Chapter 2. Instead of the standard reduction by Schaefer [Sch78], we will make use of a polynomial-time many-one reduction f from 1-3-SAT to the problem $(2, \{0,1\}, \mathbb{N})$-Partition by Heggernes and Telle [HT98]. The reduction f has the following properties for a given boolean formula φ:

$$\varphi \in \text{1-3-SAT} \implies \alpha(f(\varphi)) = 2 \tag{4.3}$$

$$\varphi \notin \text{1-3-SAT} \implies \alpha(f(\varphi)) = 3. \tag{4.4}$$

We will give a short description of reduction f, which we will use to prove the claim of Theorem 4.5. Let the boolean formula $\varphi = \varphi(X, C)$ over the variable set $X = \{x_1, x_2, \ldots, x_n\}$ be given. Recalling the remark after Definition 2.9, we may assume that all literals in each of the clauses in C are positive, thus C can be seen as a collection $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ of m sets containing exactly three elements over the variable set X. The polynomial-time many-one reduction f from the set 1-3-SAT to $(2, \{0,1\}, \mathbb{N})$-Partition that maps boolean formula φ to a graph $G = f(\varphi)$ goes as follows.

First, a 4-clique containing the vertices t_1, t_2, t_3, and s is created. Then, for each set $S_i = \{x_i, y_i, z_i\}$, there is a 4-clique C_i in G induced by the vertices x_i, y_i, z_i, and a_i. For each literal x, there is an edge e_x in G. Remember that φ contains only positive literals. For each S_i in which x occurs, both endpoints of e_x are connected to the vertex x_i in the clause C_i corresponding to $x \in S_i$. Finally, each clause element C_i is connected to vertex s by an edge between s and a_i. This completes the description of reduction f. Figure 4.2 shows the graph G which resulted from the transformation of formula $\varphi = (x \vee y \vee z) \wedge (v \vee w \vee x) \wedge (u \vee w \vee z)$ when applied to reduction f. The figure is taken from [RR06b].

Reduction f is the basis for our construction to prove DP-hardness of Exact-$(5, \{0,1\}, \mathbb{N})$-Partition. The property of our reduction g is

$$(\varphi_1 \in \text{1-3-SAT} \wedge \varphi_2 \notin \text{1-3-SAT}) \iff \alpha(g(\varphi_1, \varphi_2)) = 5 \tag{4.5}$$

for any two given boolean formulas φ_1 and φ_2 such that $\varphi_2 \in$ 1-3-SAT implies that $\varphi_1 \in$ 1-3-SAT. Once Equation (4.5) is fulfilled, we can apply Lemma 3.2 to prove that Exact-$(5, \{0,1\}, \mathbb{N})$-Partition is DP-hard.

Reduction g is derived from the presented reduction f as follows. Given the two boolean formulas φ_1 and φ_2 with disjoint variable sets, let $G_{1,1}$ and $G_{1,2}$ be two disjoint copies of the graph $f(\varphi_1)$, and let $G_{2,1}$ and $G_{2,2}$

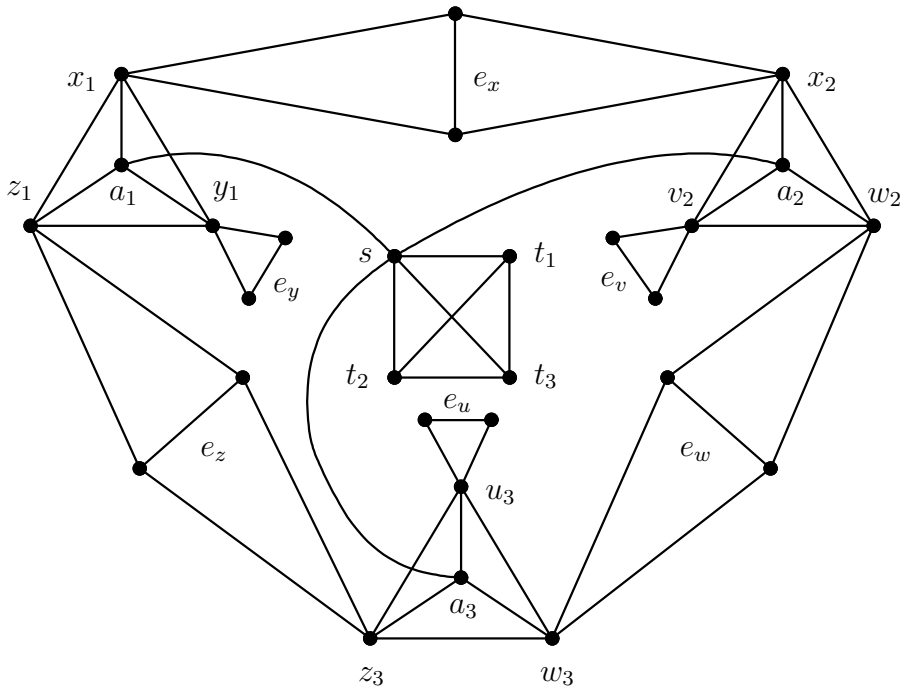

Figure 4.2: Reduction f from 1-3-SAT to the set $(2, \{0,1\}, \mathbb{N})$-Partition

be two disjoint copies of the graph $f(\varphi_2)$. For $i \in \{1,2\}$, we can then define G_i to be the disjoint union of $G_{i,1}$ and $G_{i,2}$. As the final step, we define the graph $G = g(\varphi_1, \varphi_2)$ to be the join of the graphs G_1 and G_2; see Definition 2.8 in Chapter 2. Summing up, we obtain

$$g(\varphi_1, \varphi_2) = G = G_1 \oplus G_2 = (G_{1,1} \cup G_{1,2}) \oplus (G_{2,1} \cup G_{2,2}).$$

For illustration, see Figure 4.3 which shows the graph $G = g(\varphi_1, \varphi_2)$ resulting from reduction g applied to the formulas

$$\begin{aligned} \varphi_1 &= (x \vee y \vee z) \wedge (v \vee w \vee x) \wedge (u \vee w \vee z) \quad \text{and} \\ \varphi_2 &= (c \vee d \vee e) \wedge (e \vee f \vee g) \wedge (g \vee h \vee i) \wedge (i \vee j \vee c). \end{aligned}$$

The figure is taken from [RR06b].

Let $a = \alpha(G_{1,1}) = \alpha(G_{1,2})$ and $b = \alpha(G_{2,1}) = \alpha(G_{2,2})$. For the two graphs G_1 and G_2 resulting from the disjoint union, it holds that $\alpha(G_1) = a$ and $\alpha(G_2) = b$. Clearly, if we use the same partition on $G = G_1 \oplus G_2$ as on G_1 and G_2, we can conclude that $\alpha(G) \leq a + b$. As we obtain 8-cliques from joining pairs of 4-cliques, it must the case that $\alpha(G) \geq 4$,

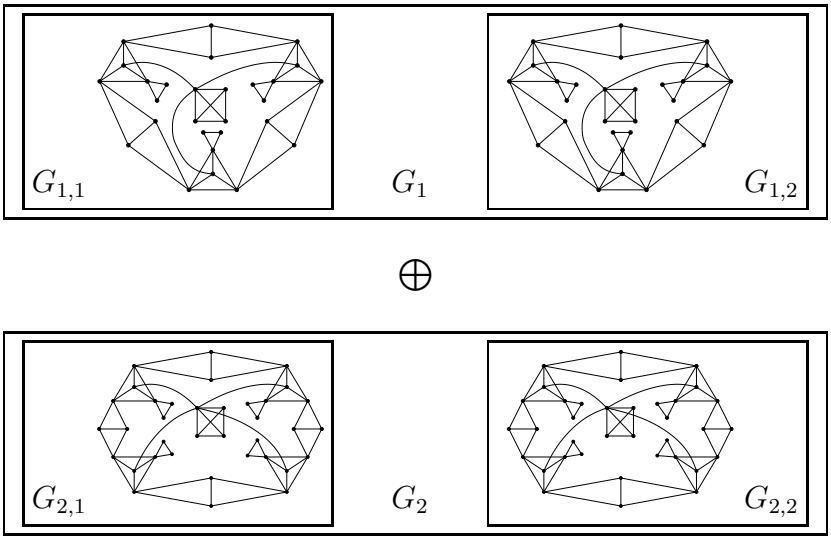

Figure 4.3: Reduction graph $G = g(\varphi_1, \varphi_2)$ from Theorem 4.5

since the vertices in each 8-clique have to be partitioned into four different $(\{0,1\},\mathbb{N})$-sets.

Our goal is to prove that $\alpha(G) = \alpha(G_1) + \alpha(G_2) = a + b$. Let $\ell = \alpha(G)$. Suppose that $\ell < a + b$. We have to distinguish three cases:

Case 1: $[\boldsymbol{a = b = 2}]$ Statement $\ell < 4$ contradicts property $\ell \geq 4$ of G, and we are done.

Case 2: $[\boldsymbol{a = 2}$ **and** $\boldsymbol{b = 3}]$ Suppose that $\ell = 4 < 5 = a + b$ and the partition of G into four $(\{0,1\},\mathbb{N})$-sets is given by V_1, V_2, V_2, and V_4. Then, one of the four disjoint $(\{0,1\},\mathbb{N})$-sets consists of at least one vertex u in G_1 and one vertex v in G_2 (Otherwise we would obtain a partition of G_1 into less than two $(\{0,1\},\mathbb{N})$-sets, or a partition of G_2 into less than three $(\{0,1\},\mathbb{N})$-sets, which contradicts Equation (4.3).) Without loss of generality, let this set be V_1. Since it is $\sigma = \{0,1\}$, and u is adjacent to every vertex in G_2, and v is connected to every vertex in G_1, we conclude for the set V_1 that $V_1 = \{u, v\}$. Then, for each of the 8-cliques in G, no valid partition into four $(\{0,1\},\mathbb{N})$-sets is possible, which is a contradiction to $\ell < 5$. It follows that $\ell = 5$.

Case 3: $[\boldsymbol{a = b = 3}]$ By the same argument used in Case 2, $\ell = 4$ cannot hold. So suppose that $\ell = 5$, and that this is witnessed by a partition

of G into the five vertex disjoint $(\{0,1\},\mathbb{N})$-sets $V_1,\ldots,V_5$. With the same argument as in Case 2, one of the sets in the partition must contain exactly one vertex u from G_1 and exactly one vertex v from G_2. Let this set be $V_1 = \{u,v\}$. The set V_1 must be the only set of the partition containing one vertex from G_1 and one vertex from G_2. Suppose otherwise, and let without loss of generality $V_2 = \{w,x\}$ be this set. Then, at least one of the 8-cliques in G does not contain any vertex from $\{u,v,w,x\}$, but a partition of the clique into three $(\{0,1\},\mathbb{N})$-sets is not possible. Therefore, every of the four remaining sets V_2, V_3, V_4, and V_5 can only have vertices from either G_1 or G_2. It follows that either two of these sets cover all vertices in G_1 except for vertex u, or two of these sets cover all vertices in G_2 except for vertex v. Suppose the first case holds, since the second case can be treated analogously. Since vertex u belongs to either $G_{1,1}$ or $G_{1,2}$, we have found a partition of the graphs $G_{1,1}$ or $G_{1,2}$ into two $(\{0,1\},\mathbb{N})$-sets. This contradicts our assumption $a = \alpha(G_{1,1}) = \alpha(G_{1,2}) = 3$. Hence, it is $\ell = 6$.

Case $a = 3$ and $b = 2$ cannot occur, since the input formulas φ_1 and φ_2 have the property that $\varphi_2 \in$ 1-3-SAT implies $\varphi_1 \in$ 1-3-SAT.

Thus we have proven $\alpha(G) = \alpha(G_1) + \alpha(G_2)$. Equation (4.5) is fulfilled and we can conclude

$$\begin{aligned} &||\{i \mid H_i \in \text{1-3-SAT}\}|| \text{ is odd} \\ &\iff H_1 \in \text{1-3-SAT} \wedge H_2 \notin \text{1-3-SAT} \\ &\iff \alpha(G_1) = 2 \wedge \alpha(G_2) = 3 \\ &\iff \alpha(G) = 5. \end{aligned}$$

Equation (3.3) of Lemma 3.2 is satisfied, and thereby we have proven the DP-completeness of Exact-$(5,\{0,1\},\mathbb{N})$-Partition.

∎

In contrast to Theorem 4.5, Exact-$(2,\{0,1\},\mathbb{N})$-Partition is in NP (and even NP-complete) and thus cannot be DP-complete unless the boolean hierarchy over NP collapses.

Theorem 4.6 Exact-$(2,\{0,1\},\mathbb{N})$-Partition *is* NP*-complete.*

Proof. Exact-$(2,\{0,1\},\mathbb{N})$-Partition can be written as

$$\text{Exact-}(2,\{0,1\},\mathbb{N})\text{-Partition} = A \cap \overline{B}$$

with $A = (2, \{0,1\}, \mathbb{N})$-`Partition` being a member of the class NP and with the set $B = (1, \{0,1\}, \mathbb{N})$-`Partition` being solvable in polynomial time. Thus, it is `Exact`-$(2, \{0,1\}, \mathbb{N})$-`Partition` $\in$ NP. The NP-hardness follows immediately via the reduction f by Heggernes and Telle defined in the proof of Theorem 4.5, see Figure 4.2:

$$\varphi \in \text{1-3-}\texttt{SAT} \iff f(\varphi) \in \texttt{Exact}\text{-}(2, \{0,1\}, \mathbb{N})\text{-}\texttt{Partition}.$$

Thus, problem `Exact`-$(2, \{0,1\}, \mathbb{N})$-`Partition` is NP-complete. ∎

The exact complexity of `Exact`-$(k, \{0,1\}, \mathbb{N})$-`Partition` is still unknown for the values $k \in \{3, 4\}$, and just as with the exact domatic number problem, the settlement of this question poses an interesting open problem.

4.3.5 The Case $\sigma = \{1\}$ and $\rho = \mathbb{N}$

The DP-completeness proof for the subcase $\sigma = \{1\}$ and $\rho = \mathbb{N}$ can be directly derived from Theorem 4.5 in the previous subsection. We will introduce another parameter for any given graph G, characterizing the number of $(\{1\}, \mathbb{N})$-sets.

Definition 4.6 *For every graph G, define the minimum value k for which G has a $(k, \{1\}, \mathbb{N})$-partition as follows:*

$$\beta(G) = \min\{k \in \mathbb{N}^+ \mid G \in (k, \{1\}, \mathbb{N})\text{-}\texttt{Partition}\}.$$

Theorem 4.7 *For each $k \geq 5$, the set* `Exact`-$(k, \{1\}, \mathbb{N})$-`Partition` *is complete for* DP.

Proof. Clearly $\alpha(G) \leq \beta(G)$ holds for all graphs $G = (V, E)$, since every partition of a graph G into $(\{1\}, \mathbb{N})$-sets induces a partition of G into $(\{0,1\}, \mathbb{N})$-sets. Conversely, we prove that it is $\alpha(G) \geq \beta(G)$. It is enough to do so for all graphs $G = f(\varphi)$ resulting from any given boolean formula φ of 1-3-`SAT` via the reduction f in Theorem 4.5.

Suppose $\varphi \in$ 1-3-`SAT`. Let $\varphi = \varphi(X, C)$. Then, we have $\alpha(G) = 2$ by the following partition into two $(\{1\}, \mathbb{N})$-sets V_1 and V_2. Let t be the satisfying truth assignment of φ in the one-in-three satisfiability sense, meaning that each clause contains exactly one literal which is set to true by t. Remember that every clause of C contains only positive literals over the variable set X. For every $x \in X$ with $t(x) = 1$, put the endpoints of the edge e_x into V_1.

For each $y \in X$ with $t(y) = 0$, put the endpoints of the edge e_y into V_2. Then, every 4-clique $\{a_i, l_1, l_2, l_3\}$ corresponding to a clause $c_i = \{l_1, l_2, l_3\}$ with $1 \leq i \leq m$ contains exactly one vertex l_j, $1 \leq j \leq 3$, that is adjacent to the endpoints of an edge e_x contained in the set V_1. Add the vertices l_j and a_i to V_2, and add the other two literal vertices l_k with $1 \leq k \leq 3$ and $k \neq j$ to the set V_1. Finally, vertex s is added to the set V_1, as is one arbitrary vertex t_j with $1 \leq j \leq 3$. The remaining vertices t_k with $1 \leq k \leq 3$ and $k \neq j$ can then be added to the set V_2. Using the same partition, we even get two $(\{1\}, \mathbb{N})$-sets for graph G, since every vertex of G has exactly one neighbor that is in the same set of the partition as the vertex itself. Hence, it is $\beta(G) = 2$.

Now suppose that $\varphi \notin$ 1-3-SAT, then it is $\alpha(G) = 3$. We can then partition G into three $(\{1\}, \mathbb{N})$-sets in the following way. V_1 consists of the vertices s and t_1 plus the endpoints of each edge e_x for $x \in X$. The set V_2 consists of t_2 and t_3, every vertex a_i for each clause $c_i \in C$, $1 \leq i \leq m$, and one more arbitrary vertex in the 4-clique c_i. The two remaining vertices in each c_i are then put into the set V_3. Thus it follows that $\alpha(G) = \beta(G)$. The rest of the proof is analogous to the proof of Theorem 4.5. ∎

In contrast to Theorem 4.7, Exact-$(2, \{1\}, \mathbb{N})$-Partition is in NP (and even NP-complete) and thus cannot be DP-complete unless the boolean hierarchy over NP collapses to its first level. The proof follows straightforward from the proofs of Theorems 4.6 and 4.7 and is omitted here.

Theorem 4.8 Exact-$(2, \{1\}, \mathbb{N})$-Partition *is* NP*-complete.*

4.4 Summary of Results

We will sum up the DP-completeness results obtained in Section 4.3 for the exact versions of partitioning graphs into generalized dominating sets. Entries marked with an asterisk are results from this thesis.

The numbers in Table 4.2 have the following meaning. A value of ∞ marks those problems Exact-(k, σ, ρ)-Partition which are solvable in polynomial time for all values $k \geq 1$. Each entry in Table 4.2 consisting of two values $i \mid j$ indicates that the corresponding minimum (maximum) problem Exact-(k, σ, ρ)-Partition is DP-complete for all $k \geq j$, and NP-complete (coNP-complete) for the value $k = i$. That is, each entry j gives the best value of k for which the problem Exact-(k, σ, ρ)-Partition is known to be

σ \ ρ	$\mathbb{N}$	$\mathbb{N}^+$
$\mathbb{N}$	∞^*	$2 \mid 5^*$
$\mathbb{N}^+$	∞^*	$1 \mid 3^*$
$\{1\}$	$2 \mid 5^*$	$-$
$\{0,1\}$	$2 \mid 5^*$	$-$
$\{0\}$	$3 \mid 4$	$-$

Table 4.2: DP-completeness for the problems `Exact`-(k, σ, ρ)-`Partition`

complete for DP. As we can see, these values are not yet optimal. For example, `Exact`-$(5, \{1\}, \mathbb{N})$-`Partition` is known to be complete for DP and `Exact`-$(2, \{1\}, \mathbb{N})$-`Partition` is known to be complete for NP. But what about `Exact`-$(t, \{1\}, \mathbb{N})$-`Partition` in the two cases $t = 3$ and $t = 4$?

Furthermore for the exact domatic number problem, the generalized version `Exact`-M_k-`DNP` could be proven BH_{2k}(NP)-complete. Unfortunately, the DP-completeness results obtained in Subsections 4.3.1, 4.3.4, and 4.3.5 cannot be generalized to the higher levels of the boolean hierarchy over NP. Again, the problem lies within the fact that it seems to be hard to perform basic mathematical operations on graph properties whenever it comes to the subject of domination.

In the next chapter we will return our focus to the domatic number problem. The NP-completeness of k-`DNP` for all $k \geq 3$ presumably negates the existence of a polynomial-time algorithm for partitioning graphs into the maximum number of vertex disjoint dominating sets—otherwise we could conclude P = NP. Therefore, our goal will be to construct exponential-time algorithms running significantly faster than the trivial brute-force algorithms that cycle through each potential solution.

Chapter 5

Exact Exponential-Time Algorithms

5.1 Overview

Many problems of practical importance have been proven NP-complete. Therefore, no efficient algorithms computing solutions to these kind of problems can exist unless P $\neq$ NP. Nevertheless, solutions need to be found in practice in some way or another. Different strategies have been developed to cope with NP-complete problems.

For example, it might suffice not to exactly solve the problem, but to approximate a valid result within a certain range. Consider set `VERTEX-COVER`, where for a given graph $G = (V, E)$ the minimum cardinality of a vertex cover $V' \subseteq V$ needs to be determined.[1] It was proven to be NP-complete by Karp [Kar72]. This problem is approximable by a factor of $1/2$, meaning that we will always find a vertex cover V'' with $\frac{1}{2} \cdot ||V''|| \geq ||V'||$, where the set V' is a vertex cover of minimum cardinality, see [GJ79]. Unfortunately for most NP-complete problems A, it can be shown that no polynomial-time approximation algorithm exists which performance ratio is better than $1-\epsilon$, where ϵ is a fixed number. Even worse, in the case of the traveling salesman problem, it is known that no polynomial-time algorithm can approximate the optimum solution, i.e. the tour with minimum distance visiting all cities, within a factor of $1 - \epsilon$ for any number $\epsilon \in (0, 1)$, see [SG76]. Then again, how do we approximate NP-complete non-optimization problems like the satisfiability problem?

[1] A subset $V' \subseteq V$ of the vertex set V of G is called a *vertex cover of* G, if at least one endpoint of each edge $e = \{u, v\}$ is contained in V'.

While most complexity classes are defined by worst-case complexity, i.e., the provided upper bound for the complexity measure is taken over all inputs of length n, one might wonder if there exist problems that are hard to solve in the worst-case, but perform well on average. This intriguing question initiated the study of average-case complexity. It was first defined by Levin [Lev86]. Given an NP-complete problem and a probability distribution on its inputs, Levin precisely defined when the problem is said to be hard to solve on average. There are two immediate advantages arising from this definition. First, any attempts to design algorithms that work fast on average appear in another light considering a proof on average hardness has been obtained. Second, in cryptographic applications, the average hardness of mathematical computations contributes to the security of the used system. On another note, problems that are NP-complete and easy to solve on average include the colorability problem [Wil84] and the Hamiltonian Path problem [GS87]. For more details on average-case complexity, we refer to the survey by Ben-David et al. [BCGL92] and Wang [Wan97].

Another strategy to deal with NP-complete problems is to design algorithms with a one-sided error. These are called randomized or probabilistic algorithms. If the error rate is sufficiently small, i.e., strictly less than $1/2$, sequentially running this method over and over leads to an overall error rate which is negligibly small. The primality problem `PRIMES` is probably the most prominent example which is handled by probabilistic algorithms in practice. Many important cryptographic applications depend on generating prime numbers, hence very fast methods to determine the primality of a random number are preferable. `PRIMES` has recently been proven to be solvable in deterministic polynomial time [AKS04]. This is an outstanding result as the complexity of `PRIMES` has long been an open question. Unfortunately, the running time of the algorithm serving as a witness for $\texttt{PRIMES} \in \mathrm{P}$ is still far away from being practical. Thus, known probabilistic methods like Miller-Rabin [Mil76, Rab80] and Solovay-Strassen [SS77] are still in use. Concerning NP-complete problems, the one-sided error rate will certainly be as high as $(1-c)^n$ for some constant $c \in (0,1)$ on an input of size n, thus the methods will need to be run sequentially an exponential number of times to lead to a marginally small overall error rate. Still, most of the time the randomized algorithms yield slightly better running times than their deterministic counterparts. In Section 5.6, two randomized algorithms for 3-`DNP` when the input is restricted to graphs with bounded maximum degree will be compared to an exact deterministic version.

Nevertheless, algorithms which might produce erroneous output are not

always desired, which leads us to the theory of exact exponential-time algorithms, where *exact* means that the optimum solution (if it exists) will be found every time the algorithm is executed. While at first glance exponential running times have to be avoided at any time, the design and analysis of such algorithms might produce results of practical importance. Consider an algorithm that has a worst-case time bound of $\tilde{\mathcal{O}}(2^n)$.[2] Even though this algorithm might only be feasible for small inputs, suppose we can improve this bound to $\tilde{\mathcal{O}}\left(\sqrt{2}^{\,n}\right) = \tilde{\mathcal{O}}(1.4143^n)$. This means that with the advanced method, we are able to handle inputs twice as large as before, as it is $\sqrt{2}^{\,2n} = 2^n$. This difference can be a decisive factor in practice. Moreover, efficient methods proposed for exponential-time algorithms might result in better time bounds for other algorithms that already work in polynomial time.

In recent years, quite a lot of different and interesting surveys have been written on the subject of exponential-time algorithms. Woeginger lists several NP-complete problems and exact exponential-time algorithms solving them with the—until then—best known worst-case time bound [Woe03]. Also, interesting open questions are posted and commented. In [Sch05], Schöning focuses on the satisfiability problem and gives details on the progress which has been made throughout the past decades to improve the worst-case time bounds for deterministic and randomized algorithms finding a satisfying truth assignment. Fomin, Grandoni and Kratsch [FGK05a] analyze a method called "measure and conquer," which takes known simple algorithms for exactly solving NP-complete problems and refining the running time analysis to obtain better worst-case time bounds. The "measure and conquer" technique goes back to Eppstein [BE05, Epp04], who used it to solve constraint satisfaction problems (CSP) and the problem 3-COLOR, which is a special case of a CSP. Finally, the survey from Riege and Rothe [RR06c] lists the progress made on deterministic and randomized algorithms solving three NP-complete problems, including the domatic number problem. Some of the algorithms for the domatic number problem are part of this thesis, see Theorems 5.2, 5.4, 5.6, 5.7, and 5.9.

[2] We use the $\tilde{\mathcal{O}}$-notation as it is common for exponential-time algorithms, since it neglects polynomial factors. For a fixed polynomial $p \in \text{poly}$ and a function g mapping from $\mathbb{N}$ to $\mathbb{N}$, it is $f \in \tilde{\mathcal{O}}(g)$ if $f \in \mathcal{O}(p \cdot g)$. To see that this definition makes sense in the exponential case, note that for an input of size $n = 10000$, the term $n^2 \cdot 1.5^n$ lies between 1.5027^n and 1.5028^n.

5.2 Partitioning by Dynamic Programming

One of the first exponential-time algorithms to solve an NP-complete problem is due to Lawler [Law76]. With a dynamic programming approach, his algorithm determines the chromatic number of any given graph G within a time bound of $\tilde{\mathcal{O}}(2.4423^n)$. His method builds on the following idea: of all legal colorings of G mapping to $\chi(G)$ values, at least one set of the partition is a maximal independent set. To see why this is true, consider any coloring f of G which maps the vertices to $\chi(G)$ independent sets, and pick one specific independent set U with $f(u) = i$ for all $u \in U$ of this partition. If U is not a maximal independent set, then there must exist a superset U' with $U \subset U'$ such that U' is a maximal independent set. Change the previous coloring f to f' by mapping all vertices in U' to the value i, and leave the rest of the coloring f untouched. Then, the partition resulting from f' into $\chi(G)$ independent sets contains the maximal independent set U'.

This idea is used to dynamically compute the chromatic number of any given graph $G = (V, E)$. Suppose we want to determine the chromatic number of a graph $G[V']$ with $V' \subseteq V$, then we can use the following formula

$$\chi(G[V']) = 1 + \min\{\chi(G[V' - V''])\},$$

where the minimum is taken over all subsets $V'' \subset V'$ such that V'' is a maximal independent set. Using an array $A[V']$ indexed with the 2^n possible subsets $V' \subseteq V$, one can dynamically compute the chromatic numbers of the graphs $G[V']$ with increasing cardinality. Note that this results in an algorithm using exponential space. The overall running time can be determined by $\text{poly}(n) \cdot \sum_{k=1}^{n} \binom{n}{k} 2^k \in \tilde{\mathcal{O}}(3^n)$. Lawler uses two results from graph theory to lower this bound even further. First, Moon and Moser [MM65] showed that there are at most $3^{n/3}$ maximal independent sets for a graph with n vertices, and second, Paull and Unger [PU59] proved that each of these sets can be generated in time $\mathcal{O}(n^2)$. Summing up, we end up with a running time of

$$\sum_{k=1}^{n} \binom{n}{k} k^2 \; 3^{k/3} \leq n^2 \; (1 + 3^{1/3})^n,$$

which leads to the algorithm computing the chromatic number of any given graph G with n vertices in time $\tilde{\mathcal{O}}(2.4423^n)$ by Lawler [Law76].

Theorem 5.1 *There exists an algorithm computing the chromatic number for any graph G with n vertices in time $\tilde{\mathcal{O}}(2.4423^n)$ and exponential space.*

One drawback of Lawler's method is that exponential space is needed to store the chromatic numbers $\chi(G[V'])$ for the subsets $V' \subseteq V$. The same dynamic programming method is used by Fomin, Grandoni, and Kratsch to compute the domatic number of any graph [FGK05a]. This algorithm will be presented in the next section.

5.3 Breaking the Trivial Barrier

We will focus our interest on the domatic number problem in the last sections of this thesis. Look at the decision version k-DNP of the domatic number problem. Given a graph $G = (V, E)$ and a positive integer k, the question is whether there exists a partition into k dominating sets, i.e., if it is $\delta(G) \geq k$. The naive way to answer this question is to sequentially skip through every possible partition of V into k sets $D_1, D_2, \ldots, D_k$ and check whether each of the sets D_i is a dominating set of G for $1 \leq i \leq k$. The number of potential solutions, i.e., the number of different ways to partition a set with n elements into k nonempty, disjoint subsets can be computed by the Stirling numbers of the second kind

$$S_2(n,k) = \frac{1}{k!}\sum_{i=0}^{k-1}(-1)^i\binom{k}{i}(k-i)^n,$$

which leads to a running time of $\tilde{\mathcal{O}}(k^n)$.

For all $k \geq 4$, an improvement of this worst-case time bound to decide the set k-DNP can be made by the dynamic programming approach for the colorability problem presented in the last section. This comes at the cost of working in exponential space. For a graph $G = (V, E)$ and a subset $V' \subseteq V$ of the vertex set, define by $\mathtt{DN}(G|V')$ the maximum number of vertex disjoint subsets of V' such that each of these sets is a dominating set of G. Then, similar to the colorability case, we can compute $\mathtt{DN}(G|V')$ if for all proper subsets $V'' \subset V'$ the values $\mathtt{DN}(G|V'')$ are known by

$$\mathtt{DN}(G|V') = 1 + \max\{\mathtt{DN}(G|V' - D)\},$$

where the maximum is taken over all minimal dominating sets $D \subseteq V'$. When no assumptions about the number of minimal dominating sets can be made, this straightforwardly leads to an algorithm with a running time of

$$\sum_{k=0}^{n}\binom{n}{k}2^k = (1+2)^n = 3^n.$$

Proposition 5.1 *For any given graph $G = (V, E)$ with n vertices, there exists an exponential-space algorithm computing the domatic number $\delta(G)$ in time $\tilde{\mathcal{O}}(3^n)$.*

Before turning to an algorithm extending and refining this idea even further by Fomin et al. [FGPS05], we concentrate on the special case of 3-DNP and the first algorithm breaking the natural 3^n barrier, which is due to Riege and Rothe [RR05]. Before we can describe their method to decide whether it is $\delta(G) \geq 3$ for a graph G, we need to define the sets which will be used in their algorithm to store all relevant data.

The general idea of the algorithm goes as follows. In the beginning of the algorithm, three empty sets D_1, D_2, and D_3 are initialized, which in the successful case will end up being dominating sets of the input graph $G = (V, E)$. In each step, one vertex not yet assigned to any of these three sets will be selected by a greedy strategy. The "usefulness" of a vertex in the process of constructing the three dominating sets is strongly related to its degree. Intuitively, the larger the degree of a vertex $v \in V$ and therefore the cardinality of $N[v]$, the more vertices are potentially dominated by v when it is added to one of the sets D_i for $1 \leq i \leq 3$. The technical notions in the next definition try to capture this (local) measure precisely.

Definition 5.1 *Let $G = (V, E)$ be a graph with n vertices, then define with $\mathcal{P} = (D_1, D_2, D_3, R)$ the partition of V into four sets, D_1, D_2, D_3, and R. Our recursive algorithm will eventually construct three dominating sets from the sets D_i of V (if they exist), and the subset $R \subseteq V$ collects the remaining vertices not yet assigned at the current point in the computation of the algorithm. With $r = ||R||$ we define the number of the remaining vertices, then $d = n - r$ is the number of vertices which have already been assigned to some set D_i. The* area of G covered by partition $\mathcal{P}$ *is defined as* $\text{area}_{\mathcal{P}}(G) = \sum_{i=1}^{3} ||N[D_i]||$. *Note that* $\text{area}_{\mathcal{P}}(G) = 3n$ *if and only if D_1, D_2, and D_3 are dominating sets of G. By defining the* surplus of graph G *as* $\text{surplus}_{\mathcal{P}}(G) = \text{area}_{\mathcal{P}}(G) - 3d$, *we want to express the progress that the algorithm has already made when it currently is in a recursive step with partition $\mathcal{P}$. To remember assignments to the sets D_1, D_2, and D_3 that did not lead to a valid solution, some of the vertices in R may be assigned to three, not necessarily disjoint, auxiliary sets A_1, A_2, and A_3. Define $\mathcal{A} = (A_1, A_2, A_3)$. For each vertex $v \in R$ and for each i with $1 \leq i \leq 3$, define the* gap of vertex v with respect to set D_i *by*

$$\text{gap}_{\mathcal{P},\mathcal{A}}(v, i) = \begin{cases} ||N[v]|| - ||\{u \in N[v] \mid N[u] \cap D_i \neq \emptyset\}|| & \text{if } v \notin A_i, \\ \bot & \text{otherwise,} \end{cases}$$

where $\perp$ is a special symbol that indicates that $\text{gap}_{\mathcal{P},\mathcal{A}}(v,i)$ is undefined for this v and i. (Our algorithm will make sure to properly handle the cases of undefined gaps.)

Additionally, given $\mathcal{P}$ and $\mathcal{A}$, define for all vertices $v \in R$:

$$\begin{aligned}
\text{maxgap}_{\mathcal{P},\mathcal{A}}(v) &= \max\{\text{gap}_{\mathcal{P},\mathcal{A}}(v,i) \mid 1 \leq i \leq 3\},\\
\text{mingap}_{\mathcal{P},\mathcal{A}}(v) &= \min\{\text{gap}_{\mathcal{P},\mathcal{A}}(v,i) \mid 1 \leq i \leq 3\},\\
\text{sumgap}_{\mathcal{P},\mathcal{A}}(v) &= \sum_{i=1}^{3} \text{gap}_{\mathcal{P},\mathcal{A}}(v,i).
\end{aligned}$$

Given G, $\mathcal{P}$, and $\mathcal{A}$, define the maximum gap of G *and the* minimum gap of G *by taking the maximum and minimum gaps over all vertices in G not yet assigned:*

$$\begin{aligned}
\text{maxgap}_{\mathcal{P},\mathcal{A}}(G) &= \max\{\text{maxgap}_{\mathcal{P},\mathcal{A}}(v) \mid v \in R\},\\
\text{mingap}_{\mathcal{P},\mathcal{A}}(G) &= \min\{\text{mingap}_{\mathcal{P},\mathcal{A}}(v) \mid v \in R\}.
\end{aligned}$$

Let $\mathcal{P}$ be given. A vertex $u \in V$ is called an open neighbor of $v \in V$ *if $u \in N[v]$ and u has not been assigned to any set D_1, D_2, or D_3 yet. A potential dominating set D_i, $1 \leq i \leq 3$, is called an* open set of $v \in V$ *if its closed neighborhood does not include v, i.e., v is not dominated by D_i. The* balance of $v \in V$ *is defined as the difference between the number of open vertices and the number of open sets. Formally, define*

$$\begin{aligned}
\text{openNeighbors}_{\mathcal{P}}(v) &= N[v] \cap R,\\
\text{openSets}_{\mathcal{P}}(v) &= \{i \in \{1,2,3\} \mid N[v] \cap D_i = \emptyset\},\\
\text{balance}_{\mathcal{P}}(v) &= ||\text{openNeighbors}_{\mathcal{P}}(v)|| - ||\text{openSets}_{\mathcal{P}}(v)||.
\end{aligned}$$

For a partition $\mathcal{P}$, a critical vertex $v \in V$ *is a vertex with a balance equal to zero and which is not dominated by all three potential dominating sets D_i where $1 \leq i \leq 3$. More formally, vertex $v \in V$ is critical if and only if it is* $\text{balance}_{\mathcal{P}}(v) \leq 0$ *and* $||\text{openSets}_{\mathcal{P}}(v)|| > 0$.

Once a vertex $v \in V$ turns critical, that is $\text{balance}_{\mathcal{P}}(v) = 0$, the vertices remaining in $N[v] \cap R$ have to be pairwise assigned to different dominating sets. Otherwise, $\text{balance}_{\mathcal{P}}(v)$ would turn negative and no valid partition can be constructed in this case anymore. This limit on the choices left to be made leads to a bound of the size of the recursion tree by a certain factor. Thus, we end up with a worst-case running time—though only marginal—below the trivial 3^n barrier.

Proposition 5.2 *Let $\mathcal{P} = (D_1, D_2, D_3, R)$ be the partition given in Definition 5.1, and let $v \in V$ be a critical vertex for $\mathcal{P}$. The only way to modify $\mathcal{P}$ so as to contain three dominating sets is to assign all vertices $u \in N[v] \cap R$ to distinct dominating sets D_i.*

The strategy of the algorithm is to recursively assign the remaining vertices $v \in R$ to the three potential dominating sets D_1, D_2, and D_3. We will always pick the vertex v with the highest maximum gap $\text{maxgap}_{\mathcal{P},\mathcal{A}}(v)$ for the current partition $\mathcal{P}$ and state of $\mathcal{A}$. Once one choice to put v into a set D_i with $1 \leq i \leq 3$ turns out to be wrong because of $\text{balance}_{\mathcal{P}}(u) < 0$ for some vertex $u \in V$, we remember this by assigning v to the set A_i. After each step, all gaps can be dynamically recalculated within polynomial time.

The following Theorem due to Riege and Rothe [RR05] describes the algorithm in detail and analyzes its running time.

Theorem 5.2 *There exists an algorithm working in polynomial-space and solving the problem* 3-DNP *in time $\tilde{\mathcal{O}}(2.9416^n)$.*

Proof. Let $G = (V, E)$ be the given graph with n vertices. Since it is $\delta(G) \leq \textit{max-deg}(G) + 1$, we may assume without loss of generality that it holds $\textit{max-deg}(G) \geq 2$, since this property can be recognized in polynomial time. The pseudo-code of our algorithm is given in detail in the appendix, see Figures A.1, A.2, A.3, A.4, and A.5. All figures are slightly modified versions from [RR05].

Figure A.1 describes the main body of the algorithm. For initialization, we set $\mathcal{P} = (\emptyset, \emptyset, \emptyset, V)$, and $\mathcal{A} = (\emptyset, \emptyset, \emptyset)$, so each potential dominating set D_i and each auxiliary set A_i, $1 \leq i \leq 3$, is set to the empty set. Next, the recursive function DOMINATE is called for the first time with graph G, partition $\mathcal{P} = (D_1, D_2, D_3, R)$ and the not necessarily vertex disjoint sets $\mathcal{A} = (A_1, A_2, A_3)$ as parameters. Note that if the first recursive call returns without a result, i.e., without producing three dominating sets, our algorithm will output a failure message.

Function DOMINATE from Figure A.2 is called with the parameters $\mathcal{P}$ and $\mathcal{A}$. This represents the situation where vertices from $V - R$ have already been assigned to the three potential dominating sets D_1, D_2, and D_3, and every $v \in A_i$ for $1 \leq i \leq 3$ indicates that during a previous assignment, it turned out that $v \in D_i$ did not lead to a valid solution. At first, all gaps for the vertices and the graph are recalculated with respect to the current partition $\mathcal{P}$ and the sets in $\mathcal{A}$, see function RECALCULATE-GAPS from Figure A.4. Also, for each vertex $v \in V$, the values $\text{openNeighbors}_{\mathcal{P}}(v)$,

openSets$_{\mathcal{P}}(v)$, and balance$_{\mathcal{P}}(v)$ are computed. All of this can be done in polynomial time.

If the sets D_1, D_2, and D_3 are already dominating sets of G, the algorithm detects this in polynomial time and terminates with the proper output. For this, the remaining vertices $v \in R$ can be added to any set D_i, so let's say we return with the three dominating sets $D_1 \cup R$, D_2, and D_3.

If no proper partition into dominating sets has been found, function HANDLE-CRITICAL-VERTEX which is called next, detects three more trivial cases, see Figure A.5.

Case 1: It is balance$_{\mathcal{P}}(v) < 0$ for some vertex $v \in V$. This means our current potential dominating sets D_1, D_2, and D_3 cannot be extended to yield dominating sets of G, since v still needs to be dominated by ||openSets$_{\mathcal{P}}(v)$|| many sets, but only ||openNeighbors$_{\mathcal{P}}(v)$|| many vertices in its closed neighborhood can be assigned to some set D_i, $1 \leq i \leq 3$. In this case we have run into a dead-end. We return to the previous level of our recursion and mark the last choice of assigning u into some set D_i for $1 \leq i \leq 3$ as wrong by adding u to the auxiliary set A_i.

Case 2: One vertex $v \in V$ is a member of two auxiliary sets A_i, A_j with $1 \leq i < j \leq 3$. This means that v has been previously assigned to the two sets D_i and D_j, and both times the recursion returned without being able to generate three dominating sets. Thus, the only possible way to extend the current partition $\mathcal{P}$ so that it includes three dominating sets of G is to assign v to the set D_k with $1 \leq k \leq 3$ and $k \notin \{i, j\}$.

Case 3: There exists a critical vertex $v \in V$. Thus it is balance$_{\mathcal{P}}(v) = 0$ and openSets$_{\mathcal{P}}(v) > 0$ and v is not yet dominated by all three potential dominating sets D_1, D_2, and D_3 of the current partition $\mathcal{P}$. We randomly select one of the vertices in $N[v] \cap R$, say u, and sequentially try to add u to all sets D_i with $1 \leq i \leq 3$ and $u \notin A_i$ by calling function ASSIGN$(G, \mathcal{P}, \mathcal{A}, u, i)$, which eventually recursively invokes function DOMINATE, see Figure A.3. Note that especially all $v \in V$ with $deg(v) = 2$ and $N[v] \cap R \neq \emptyset$ are critical right after the initialization, therefore the algorithm will start by assigning all vertices contained in $N[v]$ first.

If none of the three cases above occurs, the algorithm will greedily select the vertex $v \in R$ with the highest maximum gap. If more than one

vertex satisfies this condition, the vertex $v \in R$ with the highest value $\text{sumgap}_{\mathcal{P},\mathcal{A}}(v)$ among the vertices with the highest maximum gap will be selected. If this condition does not produce a unique vertex, we will choose one vertex $v \in R$ randomly from this set. The selected vertex v with $\text{maxgap}_{\mathcal{P},\mathcal{A}}(v) = \text{gap}_{\mathcal{P},\mathcal{A}}(v,i)$ will then be added to the set D_i, $1 \leq i \leq 3$. This is done by calling $\textsc{Assign}(G, \mathcal{P}, \mathcal{A}, u, i)$, which assigns vertex v to D_i, removes it from R, and recursively calls function Dominate. The purpose of this selection method is that we want to raise the area of G covered by partition $\mathcal{P}$ as fast as possible, for the algorithm terminates (with a positive result) whenever we reach $\text{area}_{\mathcal{P}}(G) = 3n$. Note that we will never choose to add a vertex $v \in R$ to the set D_i for $1 \leq i \leq 3$ if $v \in A_i$, as in this case $\text{gap}_{\mathcal{P},\mathcal{A}}(v,i)$ is undefined. If the recursion fails and function $\textsc{Assign}(G, \mathcal{P}, \mathcal{A}, u, i)$ returns without a valid solution, we add vertex v to the set A_i. The final step is to call $\textsc{Dominate}(G, \mathcal{P}, \mathcal{A})$ recursively with the updated sets in $\mathcal{A}$. If this call fails again, the recursion returns to its previous level. The description of the algorithm is complete, and we will now proof its correctness.

We first mention that whenever the algorithm outputs three sets D_1, D_2, and D_3, these three sets are indeed pairwise vertex disjoint dominating sets of the graph G. Therefore, it remains to show that there exists a path in the recursion tree of the algorithm that will find this specific partition. The only drop-back occurs in Case 1 of function Handle-Critical-Vertex, when there exists a vertex $v \in V$ with $\text{balance}_{\mathcal{P}}(v) < 0$ for the current partition $\mathcal{P} = (D_1, D_2, D_3, R)$. Thus, there is no way left to extend $\mathcal{P}$ to a partition $\mathcal{P}' = (D_1', D_2', D_3', \emptyset)$ with three dominating sets D_1', D_2', and D_3' of G and $D_i \subseteq D_i'$ for $1 \leq i \leq 3$. Since the algorithm eventually checks each possible partition of G, unless stopped by such a drop-back, a partition of the vertex set V into three dominating sets will be found, if it exists. Once the algorithm has skipped through all possible partitions and no three dominating sets could be found, the first call of Dominate in the main body will return without a result. Hence, the algorithm furthermore produces the correct output in the negative case that no such partition exists.

It leaves to estimate the running time of the algorithm. Recalculating the gaps with function Recalculate-Gaps takes quadratic time in n, the number of vertices of the input graph G. Also, each verification of the three conditions in function Handle-Critical-Vertex can be checked by simply looking up the values calculated by Recalculate-Gaps, as well as the selection of a suitable vertex $v \in R$ in the recursive function Dominate.

Thus, in terms of the $\tilde{\mathcal{O}}$-notation, the running time of the algorithm depends solely on the number of recursive calls to function DOMINATE. For each recursive step of the algorithm, define by $T(m)$ the worst-case number of calls to function DOMINATE of the algorithm that might still be needed to finish the computation. Here, m is the number of potential dominating sets left for all $v \in R$ regarding the current partition $\mathcal{P} = (D_1, D_2, D_3, R)$. After initialization, the value of m is equal to $3n$, as each vertex $v \in V$ may be a member of any of the three dominating sets to be constructed (if they exist).

We will first concentrate on the case that function DOMINATE is called recursively within function HANDLE-CRITICAL-VERTEX because a critical vertex $v \in V$ has been detected. Then it is $\text{balance}_{\mathcal{P}}(v) = 0$ and we call function ASSIGN (and thus DOMINATE) for $u \in N[v] \cap R$ and each i with $1 \leq i \leq 3$ and $u \notin A_i$. Note that vertex $v \in V$ will remain critical until all vertices of $N[v] \cap R$ have been distributed among the potential dominating sets D_1, D_2, and D_3. Since $\text{openSets}_{\mathcal{P}}(v) \leq 3$, the worst-case running time in this case occurs when $\text{openSets}_{\mathcal{P}}(v) = 3$. By Proposition 5.2, all vertices in $N[v] \cap R$ have to be pairwise assigned to different sets, and since there are at most six combinations left to distribute the remaining three vertices, we end up with $T(m) \leq 6T(m-6)$ in the worst-case. The term $m - 6$ is explained by the fact that at least two choices for each vertex in $N[v] \cap R$ are possible, since otherwise function HANDLE-CRITICAL-VERTEX would have ended up in Case 2 described above. With $m = 3n$, we can conclude that $T(m) \leq 6^{m/6} = 6^{n/2}$, i.e. $T(m) \in \tilde{\mathcal{O}}(2.4495^n)$. This indicates that once function HANDLE-CRITICAL-VERTEX makes the recursive call to DOMINATE, we stay well below the running time stated in this Theorem.

The interesting case occurs whenever the recursive call originates within DOMINATE itself after HANDLE-CRITICAL-VERTEX did not interrupt the computation. Then, a vertex $v \in R$ is selected with the highest maximum gap for one i with $1 \leq i \leq 3$. Two subcases might occur. If we do not make any assumptions on the value $\text{gap}_{\mathcal{P},\mathcal{A}}(v, i)$, the running time can be bounded by the following calculation. On the one hand, the selected vertex has not been added to any of the auxiliary sets A_1, A_2, and A_3. The first call to DOMINATE thus eliminates three possible sets for v. If this call fails, v is added to A_i, so there is one less choice for v in the next recursive call, making it $T(m) \leq T(m-1)$. Summing up for this subcase, it is $T(m) \leq T(m-1) + T(m-3)$. On the other hand, vertex v might already belong to one set A_j with $1 \leq j \leq 3$. Then, after the first call to DOMINATE via function ASSIGN, we eliminate two more choices for v.

If this call fails—which will happen in the worst-case—v is added to the set A_i, and DOMINATE is called recursively another time. Since in this case v is already a member of two of the auxiliary sets, namely A_i and A_j, function HANDLE-CRITICAL-VERTEX will detect v in its second if-clause and deterministically call function ASSIGN with the parameters v and k, where $k \notin \{i, j\}$. Again, we are left with two less choices for vertex v. Summing up for the second subcase, we end up with $T(m) \leq 2T(m-2)$. Suppose that both subcases occur equally often, i.e., the algorithm considers every vertex twice. Since this may happen in the worst-case, it follows that

$$T(m) \leq \frac{1}{2}(T(m-1) + T(m-3)) + \frac{1}{2}(2T(m-2)), \tag{5.1}$$

where it is $m = 3n$. Thus, with this naive analysis we end up with a running time of $T(m) \in \tilde{\mathcal{O}}(3^n)$, and we have not been able to beat the trivial time bound.

For an improvement, we have to make sure that the recursion tree of the algorithm does not reach its full depth, i.e., not all vertices will be considered twice. This is the case when HANDLE-CRITICAL-VERTEX is called for a sufficiently large portion of the vertices. In other words, we try to reach $\text{area}_{\mathcal{P}}(G) = 3n$ as fast as possible, since the algorithm finishes (with the output of three dominating sets) once this level has been achieved. Note that for every vertex $v \in R$ that is added to one of the sets D_i, $1 \leq i \leq 3$, we increase $\text{area}_{\mathcal{P}}(G)$ by $\text{gap}_{\mathcal{P},\mathcal{A}}(v, i)$, and additionally we add $(\text{gap}_{\mathcal{P},\mathcal{A}}(v, i) - 3)$ to $\text{surplus}_{\mathcal{P}}(G)$, see Definition 5.1.

Since all vertices $v \in V$ of degree two are critical and remain so until all vertices in $N[v]$ have been removed from R, they and their neighbors can be handled in time $\tilde{\mathcal{O}}(2.4495^n)$, as argued above. So assume that it holds *min-deg*$(G) \geq 3$. Then, after initialization, we can conclude that it is $\text{maxgap}_{\mathcal{P},\mathcal{A}}(G) = \textit{max-deg}(G) + 1 > 3$. If this condition holds for $3n/4$ steps, we have reached $\text{area}_{\mathcal{P}}(G) = 3n$, and the algorithm terminates successfully. Thus, assume that at one point of the computation, $\text{maxgap}_{\mathcal{P},\mathcal{A}}(G)$ drops below four. We will exploit the fact that each level $\text{maxgap}_{\mathcal{P},\mathcal{A}}(G) = k$ can only hold for a certain portion of the remaining vertices until the maximum gap of G decreases by at least one. First we will analyze the trivial case when $\text{maxgap}_{\mathcal{P},\mathcal{A}}(G) = 0$.

Suppose $\text{maxgap}_{\mathcal{P},\mathcal{A}}(G) = 0$. Then, the recursion will stop immediately. Either we have found three disjoint dominating sets (in which case the algorithm adds the vertices in R to D_1 and halts), or at least one vertex $v \in V$ is not dominated by one of the sets D_i of the current partition $\mathcal{P}$, $1 \leq i \leq 3$. In the second case, since there exist no positive gaps

for any $v \in R$ and i with $1 \leq i \leq 3$, we are not able to extend $\mathcal{P}$ to a partition into three disjoint dominating sets. This will be detected by function HANDLE-CRITICAL-VERTEX, as it holds $\text{balance}_{\mathcal{P}}(v) < 0$ for at least one vertex $v \in V$. Thus, the algorithm will drop back one recursion level. How many vertices may we expect to be left until we reach a level with $\text{maxgap}_{\mathcal{P},\mathcal{A}}(G) = 0$? The following lemma will be helpful in estimating the size of R at this point of the computation.

Lemma 5.1 *Let $G = (V, E)$ be a graph and let $\mathcal{P} = (D_1, D_2, D_3, R)$ be a partition as in Definition 5.1. Let the number of remaining vertices be given by $r = ||R||$, and let $\text{maxgap}_{\mathcal{P},\mathcal{A}}(G) = 3$. Then it follows for at least $r/64$ vertices in R that function* DOMINATE *will not be called recursively by the algorithm.*

Proof of Lemma 5.1. Assume that $\text{maxgap}_{\mathcal{P},\mathcal{A}}(G) = k$ and that it is $k > 0$. This implies $\text{gap}_{\mathcal{P},\mathcal{A}}(v, i) \leq 3$ for each $v \in R$ and i, $1 \leq i \leq 3$, which leads to the bound $\sum_{v \in R} \text{sumgap}_{\mathcal{P},\mathcal{A}}(v) \leq 3kr$. During the next steps, each vertex selected for a set D_i with $\text{gap}_{\mathcal{P},\mathcal{A}}(v, i) = 3$ decreases at least k gaps of the vertices in $R - \{v\}$ by one. Otherwise, function HANDLE-CRITICAL-VERTEX would have found a critical vertex $u \in N[v]$ with $N[u] \cap R = \{v\}$. If $||\text{openSets}_{\mathcal{P}}(u)|| > 1$, the algorithm drops back within the recursion since condition $\text{balance}_{\mathcal{P}}(u) < 0$ is satisfied. In the case of $||\text{openSets}_{\mathcal{P}}(u)|| = 1$, the algorithm proceeds by adding v to the set D_i with $i = \text{openSets}_{\mathcal{P}}(u)$ automatically, without further branching of function DOMINATE. Thus we may assume that if no critical vertex is detected, selecting a vertex $v \in R$ for some set D_i decreases k gaps for vertices in $R - \{v\}$, and also all gaps previously defined for v are now undefined. So the lowest possible rate at which the gaps are decreased is directly related to the current maximum gap of G.

Now let us suppose $\text{maxgap}_{\mathcal{P},\mathcal{A}}(G) = 3$ and $\text{sumgap}_{\mathcal{P},\mathcal{A}}(G) = 9$ for all of the remaining vertices $v \in R$ of the current partition $\mathcal{P}$, since this is the worst-case which can occur. Remember the comment from above, that when selecting a vertex v for a set D_i with $\text{gap}_{\mathcal{P},\mathcal{A}}(v, i) = 3$, $1 \leq i \leq 3$, at least three other gaps of vertices in $R - \{v\}$ will be decreased by one. Without the occurance of critical vertices, it will take at least $r/4$ selections of vertices $v \in R$ with $\text{sumgap}_{\mathcal{P},\mathcal{A}}(v) = 9$ until we reach $\text{sumgap}_{\mathcal{P},\mathcal{A}}(v) < 9$ for all remaining vertices $v \in R$. Another $1/4$ of the $3r/4$ vertices remaining have to be added to the potential dominating sets until $\text{sumgap}_{\mathcal{P},\mathcal{A}}(v) < 8$ for all $v \in R$. Selecting another $1/4$ of the $9r/16$ vertices left in R, we have reached $\text{maxgap}_{\mathcal{P},\mathcal{A}}(G) = 2$ and it is $\text{sumgap}_{\mathcal{P},\mathcal{A}}(v) = 6$ for all $v \in R$.

This implies that every defined gap is equal to two. Summing up, we have removed

$$\frac{1}{4} \cdot r + \frac{1}{4} \cdot \frac{3}{4}r + \frac{1}{4} \cdot \frac{9}{16}r = \frac{37}{64}r$$

vertices from the set R until $\text{maxgap}_{\mathcal{P},\mathcal{A}}(G) = 2$, under the condition that a minimum number of gaps is reduced in each step, while simultaneously trying to reduce the maximum summation gap in the fastest possible way. This is the worst-case that can happen, i.e., level $\text{maxgap}_{\mathcal{P},\mathcal{A}}(G) = 0$ is reached with as few vertices left in R as possible.

With the same argument, we can show that level $\text{maxgap}_{\mathcal{P},\mathcal{A}}(G) = 2$ can hold for another $19r/64$ vertices in R. It takes $7r/64$ more vertices in R to drop from $\text{maxgap}_{\mathcal{P},\mathcal{A}}(G) = 1$ to the stage $\text{maxgap}_{\mathcal{P},\mathcal{A}}(G) = 0$. This leaves us with $r/64$ vertices in R which do not have to be processed recursively by the algorithm. ∎ Lemma 5.1

Continuing the proof of Theorem 5.2, note that we assumed for the input graph that *min-deg*$(G) \geq 3$ at the start of the algorithm. As long as condition $\text{maxgap}_{\mathcal{P},\mathcal{A}}(G) > 3$ is true, the surplus of G, which was initialized with $\text{surplus}_{\mathcal{P}}(G) = 0$, is increasing. It will decrease whenever one of the remaining vertices $v \in R$ is added to one of the sets D_i, $1 \leq i \leq 3$, with $\text{gap}_{\mathcal{P},\mathcal{A}}(v, i) < 3$. The surplus decreases by one for $\text{gap}_{\mathcal{P},\mathcal{A}}(v, i) = 2$, and it decreases by two if it is $\text{gap}_{\mathcal{P},\mathcal{A}}(v, i) = 1$.

At one state of the algorithm, let $S = \text{surplus}_{\mathcal{P}}(G)$ be the surplus collected until we reached partition $\mathcal{P}$ with $\text{maxgap}_{\mathcal{P},\mathcal{A}}(G) = 3$. The worst-case, i.e., the most number of recursive branching into function DOMINATE, will occur when we even out the surplus completely at the end of one branch of the recursion tree. For this to happen, there have to be at least $r = ||R||$ vertices left with

$$0 \cdot \frac{37r}{64} + 1 \cdot \frac{19r}{64} + 2 \cdot \frac{7r}{64} = S,$$

and we get $r \geq 64S/33$. As Lemma 5.1 states, a fraction of $1/64$ of these vertices will not be handled with recursive branching, which in this case is at least $S/33$.

The remaining question to answer is how big the surplus might grow and how many vertices are left in R once we reach a partition $\mathcal{P}$ such that $\text{maxgap}_{\mathcal{P},\mathcal{A}}(G) = 3$. The worst-case, that is the lowest surplus with as few vertices left in R as possible occurs when the input graph G satisfies *max-deg*(G) = *min-deg*$(G) = 3$. The surplus S is increased by one in each step when adding a vertex $v \in R$ to a set D_i, $1 \leq i \leq 3$ with $\text{gap}_{\mathcal{P},\mathcal{A}}(v, i) = 4$ until $\text{maxgap}_{\mathcal{P},\mathcal{A}}(G) = 3$. So selecting vertex $v \in R$ with

degree three for a set D_i, the gap of the vertices $u \in N(v)$ in its open neighborhood and the gaps of the neighbors of every u might be decreased. Summing up, at most $1+3+3\cdot 2 = 10$ vertices decrease their gaps for some i in this step of the computation. So it takes at least $n/10$ recursive branching steps for the algorithm to reach a partition $\mathcal{P}$ with $\text{mingap}_{\mathcal{P},\mathcal{A}}(G) = 3$ in the worst case. For the next branching steps, we cannot be sure that the selected vertex $v \in R$ with the highest maximum gap for a set D_i satisfies $\text{gap}_{\mathcal{P},\mathcal{A}}(v,i) > 3$. What we know is that the surplus S collected so far is at least $3n/10$, thus we obtain the two inequalities $64n/110 \leq r \leq 7n/10$. The first inequality tells us that for at least $n/110$ vertices remaining in R for the current partition $\mathcal{P}$, we never branch into two recursive calls of function DOMINATE. Setting $m = 3(109n/110)$ in Equation (5.1), the overall worst-case running time of the presented algorithm is $\tilde{\mathcal{O}}(2.9416^n)$.

∎

Note that the above algorithm uses polynomial space. Additionally to solving the decision problem 3-DNP, our method always produces three vertex disjoint dominating sets (if they exist). While the algorithm from Theorem 5.2 marks the first step in breaking the trivial 3^n brute-force method, its running time lies only slightly below the trivial barrier. Much better worst-case time bounds will be presented in the next section.

5.4 The Measure & Conquer Technique

Most exact exponential-time algorithms for NP-complete problems are of recursive nature. Starting with an instance of the problem, in each step they either try to simplify the given input directly or branch into two or more cases which then have to handle smaller problem instances. In the analysis of the worst-case running time of such algorithms, the number of nodes in the search tree of the algorithm is upper bounded and measured with respect to some parameter of the input, for example the number of variables in the case of the satisfiability problem. One small example will follow.

Take the NP-complete problem 3-SAT with input formula $\varphi = \varphi(X, C)$ over n variables. The trivial algorithm skipping through each possible assignment t and verifying if indeed t is a satisfying truth assignment runs in time $\tilde{\mathcal{O}}(2^n)$. A simple idea by Monien and Speckenmeyer improves this bound considerably [MS85]. In each recursive step, randomly pick a clause $c = (l_1, l_2, l_3)$. Now it suffices to recursively branch into the

cases $\varphi_{|l_1=1}$, $\varphi_{|l_1=0,l_2=1}$, and $\varphi_{|l_1=l_2=0,l_3=1}$.[3] Thus, the algorithm branches into three cases. If the size of the boolean formula is measured by its number of variables, the algorithm therefore continues with boolean formulas having one, two, and three less variables then before. This leads to the recursion

$$T(n) \leq T(n-1) + T(n-2) + T(n-3),$$

where $T(n)$ is the number of steps the algorithm has to make in the worst case when handling input of size n. This recurrence can be dissolved to the term $T(n) = \alpha^n$, where $\alpha = 1.8393$ is the largest real root of the equation $\alpha^3 - \alpha^2 - \alpha = 0$. Starting with this approach and using another proposition regarding autark partial assignments,[4] Monien and Speckenmeyer [MS85] were able to push the time bound of their exponential-time algorithm even further to $\tilde{\mathcal{O}}(1.6181^n)$.

Further refinements of the branching steps led to better time bounds, at the cost of having to deal with a rather complicated worst-case analysis. Eppstein lower bounded the progress of recursive algorithms with a whole new approach [Epp03, Epp04]. Instead of designing more sophisticated methods with dozens of subcases that can occur, Eppstein took well-known simple backtracking procedures and improved their worst-case time bound by a more careful choice of the measure of a problem instance. Surprisingly, much better running times could be obtained by this new method. For example, Beigel and Eppstein [BE05] constructed the currently fastest exact algorithm for 3-COLOR that has a running time of $\tilde{\mathcal{O}}(1.3289^n)$.

Fomin, Grandoni, and Kratsch called Eppstein's new method "measure and conquer," and they provided a new time bound for a known algorithm solving the minimum dominating set problem [FGK05a]. For any graph G with n vertices, they improve the bound of one and the same algorithm from $\tilde{\mathcal{O}}(1.8026^n)$ to $\tilde{\mathcal{O}}(1.5137^n)$, only redefining the measure on the given input. Instead of simply using n, the number of vertices in the given graph,

[3] Given a boolean formula $\varphi = \varphi(X, C)$, we call t' a *partial truth assignment* of the variable set X if not all variables in X have been mapped to one of the values 0 and 1, respectively. Then, by $\varphi_{|t'}$ we denote the formula which results from applying the partial assignment t' to φ: every clause $c \in C$ containing a literal l with $t'(l) = 1$ is eliminated, since it is satisfied by t'. Also, each literal l with $t'(l)$ is removed from each clause. If, during such a step, we end up with an empty clause c, assignment t' cannot be extended to any satisfying truth assignment of φ. Otherwise, if we end up with an empty clause set $C = \emptyset$, each extension of t' is a satisfying truth assignment for φ.

[4] A partial assignment t' of a boolean formula $\varphi = \varphi(X, C)$ is called *autark* for φ, if all clauses containing at least one literal set by t' are satisfied by t'. Autarkness can be recognized in polynomial time, and if φ is satisfiable, then so is $\varphi_{|t'}$.

they define different weights for each vertex depending on the number of its neighbors. Continuing this work, Fomin et al. bound the number of minimal dominating sets for a graph with n vertices [FGPS05]. Their result is stated in the following lemma.

Lemma 5.2 *There exists an algorithm which, given as input a graph G with n vertices, lists all minimal dominating sets of G in time $\tilde{\mathcal{O}}(1.7697^n)$ and in polynomial space.*

Combining the dynamic programming approach by Lawler [Law76] and the result from Lemma 5.2, Fomin et al. derive a method for computing the domatic number of any given graph [FGPS05]. Unfortunately, their result uses an array indexed by the 2^n possible subsets of the vertex set, and thus their algorithm works in exponential space.

Theorem 5.3 *The domatic number of any given graph G with n vertices can be computed in time $\tilde{\mathcal{O}}(2.8805^n)$ and exponential space.*

The article [FGK05b] by Fomin, Grandoni, and Kratsch resumes the progress which has been made using "measure and conquer" over the last years. Additionally, they discuss exponential lower bounds for algorithms. These can be seen as an indicator of the distance between the running time analysis and optimal results. Note that an exponential lower bound for an algorithm solving an NP-complete problem does not imply that P $\neq$ NP, but rather yields a limit for the efficiency of that specific algorithm.

5.5 Improved Result for 3-DNP

The algorithm presented next exactly solves the problem 3-DNP in polynomial space with the currently best time bound in the worst-case. The following result is due to Riege, Rothe, Spakowski, and Yamamoto [RRSY06a].

Theorem 5.4 *There exists a polynomial space algorithm solving* 3-DNP *in time $\tilde{\mathcal{O}}(2.6949^n)$.*

Proof. Consider as input graph a $G = (V, E)$. Remember the remark made previously that at least one partition into three dominating sets (if it exists) contains at least one minimal dominating set. With the algorithm from Lemma 5.2, we can generate all minimal dominating sets D of G sequentially in polynomial space and time $\tilde{\mathcal{O}}(1.7697^n)$.

Given such a dominating set $D \subseteq V$, we are left with the challenge of distributing the vertices $v \in V - D$ among two sets D_0 and D_1 such that both sets are dominating sets of G as well. This problem can be reformulated as an instance to the NP-complete problem NAE-3-SAT as follows. For each minimal dominating set D found, create a boolean formula $\varphi_D = \varphi_D(X, C)$ by defining

- the variable set $X = \{x_v \mid v \in V - D\}$,
- for each vertex $v \in V$, it is

$$C_v = \left\{ \bigcup_{u \in N[V], u \notin D} x_u \right\}.$$

Thus, the clause set is defined by $C = \{C_v \mid v \in V\}$.

First we prove that it holds

$$\begin{aligned} & G \in \text{3-DNP} \\ \iff\ & \varphi_D \in \text{NAE-3-SAT for some} \\ & \quad \text{minimal dominating set } D \text{ of } G\ . \end{aligned} \tag{5.2}$$

From the direction from left to right, suppose $G \in$ 3-DNP. From the comment above, one of the partitions of V into three dominating sets of G contains a minimal dominating set D. Let the other two dominating sets of G be D_0 and D_1. For each vertex $v \in V$, the closed neighborhood $N[v]$ must contain one vertex from D_0 and one vertex from D_1. We obtain an satisfying truth assignment t in the not-all-equal sense for φ_D by setting $t(x_v) = 0$ for each $x_v \in D_0$ and $t(x_v) = 1$ for each $x_v \in D_1$.

From the direction from right to left, let $\varphi_D \in$ NAE-3-SAT for some minimal dominating set D via the satisfying truth assignment t in the not-all-equal sense. We can construct two more dominating sets D_0 and D_1 of graph G by setting

$$\begin{aligned} D_0 &= \{v \in V - D \mid t(x_v) = 0\}, \text{ and} \\ D_1 &= \{v \in V - D \mid t(x_v) = 1\}. \end{aligned}$$

Every clause must contain both true and false literals, therefore we conclude $G \in$ 3-DNP by the three vertex disjoint dominating sets D, D_0, and D_1. Thus, Equation (5.2) is true.

It leaves to find a solution to the instance φ_D of the NP-complete problem NAE-3-SAT. Notice that φ_D consists of exactly n clauses, one for each vertex $v \in V$. First we give an easy polynomial-time many-one reduction from NAE-3-SAT to SAT by doubling the number of clauses. This is the fastest method known to the author to solve an instance of the NP-complete NAE-3-SAT problem. For each clause $C_v = (l_1, \vee \ldots, l_k)$, add the clause $C_{\overline{v}} = (\overline{l_1}, \vee \ldots, \overline{l_k})$, where $\overline{l_i}$ denotes the negation of the literal l_i. Define the clause set $C^- = \{C_{\overline{v}} \mid v \in V\}$, then construct the boolean formula $\varphi'_D = \varphi'_D(X, C \cup C^-)$ which has the property

$$\varphi_D \in \texttt{NAE-3-SAT} \Longleftrightarrow \varphi'_D \in \texttt{SAT}.$$

Note that the number of literals in each clause is only bounded by the maximum degree of graph G, which might be as large as $n-1$. Therefore, we will make use of an algorithm for the satisfiability problem where the running time depends on the number of clauses instead of the number of variables, as is the case with most known algorithms. The algorithm with the up-to-date best known such time bound and working in polynomial space is due to Yamamoto [Yam05]. Slightly modifying an algorithm by Hirsch [Hir00] and refining the runtime analysis, Yamamoto provides an $\tilde{\mathcal{O}}(1.234^m)$ time bound, where m is the number of clauses of the boolean formula considered. By construction, the formula φ'_D consists of exactly $2n$ clauses. Hence, the overall running time of our algorithm deciding the set 3-DNP is

$$\tilde{\mathcal{O}}(1.7697^n \cdot 1.234^{2n}) = \tilde{\mathcal{O}}(2.6949^n).$$

The algorithm uses polynomial space, as both the method listing all minimal dominating sets and Yamamoto's procedure to solve the satisfiability problem run in polynomial space with respect to the input size.

∎

A recent result by Björklund and Husfeldt beats the worst-case time bound of Theorem 5.4 at the cost of working in exponential space [BH06]. They also give a polynomial-space algorithm, which curiously has the same time bound as the algorithm of Fomin et al. [FGPS05].

Theorem 5.5 *There exists an exponential-space algorithm for computing the domatic number of a given graph that runs in time $\tilde{\mathcal{O}}(2^n)$. There exists a polynomial-space algorithm for this problem running in time $\tilde{\mathcal{O}}(2.8805^n)$.*

Note that the related special case 3-COLOR of the colorability problem can be solved much easier, i.e. in less time in the worst-case, than the three domatic number problem. Analogously to 3-DNP, where the first step was to find a suitable minimal dominating set, an algorithm deciding whether a graph $G = (V, E)$ is colorable with three colors might start with the first set of the partition by listing all maximal independent sets. The difference and advantage of the colorability problem compared to the domatic number problem lies within the fact that the remaining problem is solvable in polynomial time. Given a maximal independent set I for G, it is $\chi(G) \leq 3$ if the graph $G[V - I]$ is bipartite. Thus the exponential running time in the $\tilde{\mathcal{O}}$-notation solely depends on the first listing problem. Lawler [Law76] noted this first and gave an $\tilde{\mathcal{O}}(1.4422^n)$ algorithm for 3-COLOR, which is far better than the fastest exact algorithm solving 3-DNP.

The apparent "hardness" of dominance when compared to independence can be reflected in the related problems of finding a minimum dominating set and a maximum independent set. While the best worst-case time bound for the first mentioned problem currently lies at $\tilde{\mathcal{O}}(1.5137^n)$, much faster exact algorithms are known to solve the second problem, with the currently best time bound of $\tilde{\mathcal{O}}(1.1889^n)$. The first result for the minimum dominating set problem is due to Fomin, Grandoni, and Kratsch [FGK05a]; the second algorithm regarding the maximum independent set problem was constructed by Robson [Rob01]. As mentioned before, the global property of dominance may be the decisive factor for the higher time bounds, as for verifying that a set $D \subseteq V$ is dominating graph G, each vertex $v \in V$ has to be checked. In contrast, a set $I \subseteq V$ is an independent set whenever no two vertices *in the set* I share an edge, so this check can be restricted to the vertices in I.

5.6 Randomized Algorithms

The last section of this thesis deals with the domatic number problem with a bound on the maximum degree of the input graphs. Naturally, there are less potential solutions available to partition a graph into three dominating sets for small values of the maximum degree. Theorem 5.6 and 5.7 are due to Riege and Rothe [RR05], while Theorem 5.9 was obtained jointly with Rothe, Spakowski, and Yamamoto [RRSY06a].

For the three domatic number problem, let $G = (V, E)$ be the given graph with maximum degree $\Delta = \textit{max-deg}(G)$. The case $\Delta = 2$ is trivially solvable. Graphs with this property consist of components which are ei-

ther isolated vertices, paths, or cycles, and each of these properties can be recognized in polynomial time.

Proposition 5.3 *For any graph $G = (V, E)$ with maximum degree $\Delta = 2$, there exists a partition of V into three vertex disjoint dominating sets if and only if every component of G is a cycle of length k such that 3 divides k.*

For the algorithms in Theorems 5.6 and 5.7, we use the notation from Definition 5.1 of Section 5.3 to describe a snapshot within the algorithm. For input graph $G = (V, E)$, let $\mathcal{P} = (D_1, D_2, D_3, R)$ be a partition of the vertex set V during the process of the algorithms, where D_1, D_2, and D_3 are the potential dominating sets which need to be constructed, and the set R contains all remaining vertices which have not been assigned to one of these three sets. We will not need the auxiliary sets $\mathcal{A} = (A_1, A_2, A_3)$ used in the description of the algorithm in Theorem 5.2. We will restrict the input to connected graphs only, as each component can be recognized and treated separately within the same time bounds.

Theorem 5.6 *For a given graph G with maximum degree Δ, there exists a deterministic algorithm solving* 3-DNP *in time $\tilde{\mathcal{O}}(d^{\frac{n}{\Delta}})$, where*

$$d = \sum_{a=0}^{\Delta-2} \left[\binom{\Delta}{a} \sum_{b=1}^{\Delta-a-1} \binom{\Delta - a}{b} \right]. \tag{5.3}$$

Proof. We will construct an algorithm with the above worst-case running time as follows. First, we pick a vertex $v \in V$ at random and assign it to the first potential dominating set D_1. In each step, we check in polynomial time whether the partition $\mathcal{P} = (D_1, D_2, D_3, R)$ already consists of three dominating sets. If this is not the case, a vertex $v \in R$ is picked which is not yet dominated by all three sets and such that there exists a vertex $u \in N[v]$ in its closed neighborhood which has already been assigned to one of the sets D_i, where $1 \leq i \leq 3$. Clearly, it holds that $1 \leq ||\text{openSets}_{\mathcal{P}}(v)|| \leq 2$.

If $\text{balance}_{\mathcal{P}}(v) < 0$, we drop back within the recursion. Otherwise, all possible combinations to partition the vertices of $N[v] \cap R$ into the sets D_1, D_2 and D_3 are tried, except the ones that would lead to vertex v not being dominated by all three sets. If no such partition leads to the goal of constructing three dominating sets for G, we again drop back in the recursion.

Now the question about the worst-case running time of the above algorithm remains. The case with the most choices left occurs when we select

a vertex $v \in V$ with the property $\text{balance}_{\mathcal{P}}(v) \geq 0$, $||\text{openSets}_{\mathcal{P}}(v)|| = 2$, and $||N[v] \cap (D_1 \cup D_2 \cup D_3)|| = 1$. Suppose $N[v] \cap D_1 = u$, so vertex v is already dominated by the set D_1. We still have to assign at least one vertex in $N[v] \cap R$ to the set D_2, and at least one vertex in $N[v] \cap R$ to the set D_3. The lower the degree of vertex v in graph G, the less choices to partition the remaining vertices in $N[v] \cap R$ are left. So having a small bound on the maximum degree of graph G leads to better running times.

More precisely, for graph G with maximum degree $\Delta = \textit{max-deg}(G)$ and for vertex v picked above, any number between 0 and $\Delta - 2$ of vertices in $N[v] \cap R$ are allowed to be assigned to set D_1. If we call this number a, any number between 1 and $\Delta - a - 1$ of vertices in $N[v] \cap R$ may be chosen for the potential dominating set D_2. After these two choices have been made, assign the remaining vertices to the set D_3. This calculation leads to Equation (5.3), and since exactly Δ vertices have been transferred from R to the three potential dominating sets D_1, D_2, and D_3, we end up with a worst-case running time of $\tilde{\mathcal{O}}(d^{\frac{n}{\Delta}})$.

∎

The specific numbers are listed in Table 5.1 for graphs G with maximum degree Δ, where $3 \leq \Delta \leq 9$.

Next we present an algorithm using randomization to speed up the running time to find three vertex disjoint dominating sets. The procedure is similar to the deterministic algorithm described above, with the difference that when picking a vertex $v \in V$, not all vertices in $N[v] \cap R$ will be distributed among the sets D_i with $1 \leq i \leq 3$. Instead, one or two vertices will be assigned to the open sets at random such that vertex v is dominated by the three potential dominating sets after each probabilistic move.

Theorem 5.7 *Let G be a given graph with maximum degree Δ. Define the value d as in Equation (5.3) in Theorem 5.6. Then, for each constant $c > 0$, there exists a randomized algorithm solving* 3-DNP *with error probability at most e^{-c} in time $\tilde{\mathcal{O}}(r^{\frac{n}{2}})$, where it is*

$$r = \frac{d}{3^{\Delta-2}} \ . \tag{5.4}$$

Proof. First we describe how the randomized algorithm works. Given is a graph $G = (V, E)$ with maximum degree $\textit{max-deg}(G) = \Delta$. Just as the deterministic algorithm from Theorem 5.6, we start by randomly selecting a vertex $v \in V$ and assigning it to the potential dominating set D_1 of the partition $\mathcal{P} = (D_1, D_2, D_3, R)$.

In each following step, we select a vertex $v \in V$ that is already dominated by at least one but not all of the potential dominating sets D_1, D_2, and D_3. Therefore it is $1 \leq ||\text{openSets}_{\mathcal{P}}(v)|| \leq 2$. Distinguish the following two cases. If $||\text{openSets}_{\mathcal{P}}(v)|| = 1$, it is $N[v] \cap D_i = \emptyset$ for one i with $1 \leq i \leq 3$. Randomly select a vertex from $N[v] \cap R$ and add it to the set D_i. In the other case where $||\text{openSets}_{\mathcal{P}}(v)|| = 2$, randomly select two vertices from $N[v] \cap R$, which will then be put into the two potential dominating sets which previously did not contain a vertex from the closed neighborhood $N[v]$ of v. In this case, yet another probabilistic move has to be made as how to distribute the two random vertices into the two open sets of v.

The success probability of the algorithm depends on the error rate in each step of the algorithm. Suppose that G indeed can be partitioned into three disjoint dominating sets. In each step, the highest error rate can occur when $||N[v] \cap (D_1 \cup D_2 \cup D_3) = 1||$ and $deg(v) = \Delta$. We can restrict ourselves to this case, as we only want to supply a worst-case bound. To end up with a valid partition into three dominating sets, there are at most d choices left to distribute the vertices in $N[v] \cap R$ among the sets D_1, D_2, and D_3. Here, the number d is from Equation (5.3), and how this number is obtained is described in the proof of Theorem 5.6. After making our random choices, we have $\Delta - 2$ vertices remaining in $N[v] \cap R$, so we are left with $3^{\Delta-2}$ possibilities for a partition. Thus, the success rate in each step can be expressed by $3^{\Delta-2}/d$.

We will use the standard method to achieve an error rate below e^{-c} for each constant $c > 0$. This is done by repeating the above algorithm for a certain number of rounds. The number of repetitions is equal to the reciprocal of the success rate in each turn, which explains the value for r in Equation (5.4). Since we process two vertices in each step of the algorithm, we end up with a worst-case bound of $\tilde{\mathcal{O}}(r^{n/2})$.

∎

The specific running times are listed in Table 5.1 for graphs G with maximum degree Δ, where $3 \leq \Delta \leq 9$.

The last algorithm of this chapter is derived from Schöning's method to solve constraint satisfaction problems, or `CSP` in short [Sch99]. Schöning proved the following theorem.

Theorem 5.8 *For each of the values $a > 2$, $b \geq 2$, and any $\epsilon > 0$, there exists a polynomial-space algorithm for the problem (a, b)-`CSP` running in time*

$$\tilde{\mathcal{O}}(a(1 - \tfrac{1}{b}) + \epsilon).$$

Δ	3	4	5	6	7	8	9
Thm. 5.4	2.695^n	2.695^n	2.695^n	2.695^n	2.695^n	2.695^n	2.695^n
Thm. 5.6	2.290^n	2.660^n	2.826^n	2.906^n	2.948^n	2.970^n	2.983^n
Thm. 5.7	2^n	2.358^n	2.583^n	2.727^n	2.820^n	2.881^n	2.922^n
Thm. 5.9	2.251^n	2.401^n	2.501^n	2.572^n	2.626^n	2.667^n	2.701^n

Table 5.1: Running times of the algorithms for bounded-degree graphs

This directly leads to the randomized algorithm for the domatic number problem described below which is due to Riege, Rothe, Spakowski, and Yamamoto [RRSY06a].

Theorem 5.9 *Let G be a given graph with maximum degree Δ. Then, for any $\epsilon > 0$, there exists a randomized polynomial-space algorithm solving problem* 3-DNP *with running time*

$$\tilde{\mathcal{O}}(3(1 - \tfrac{1}{\Delta+1}) + \epsilon).$$

Proof. For a graph $G = (V, E)$, we will formulate 3-DNP as a constraint satisfaction problem F with $n = ||V||$ variables which are restricted to three values, and the constraints consist of at most $\Delta + 1$ variables. Denote by $D = \{0, 1, 2\}$ the domain of the CSP instance, then the construction of F goes as follows:

- Define the variable set by $X = \{x_v \mid v \in V\}$.
- For each vertex $v \in V$, define the constraints $C_{v,i}$, $1 \leq i \leq 3$, by

$$C_{v,i}(x_v, x_{w_1}, x_{w_2}, \ldots, x_{w_{N(v)}}) = 1 \Longleftrightarrow$$
$$\text{at least one of } x_v, x_{w_1}, x_{w_2}, \ldots, x_{w_{N(v)}} \text{ is set to the value } i.$$

The order in each constraint, i.e., the number of variables it involves, is restricted by $\Delta + 1$. Applying the algorithm of Theorem 5.8 with the values $a = 3$ and $b = \Delta + 1$ we can solve this (a, b)-CSP instance in time $\tilde{\mathcal{O}}(3(1 - \frac{1}{\Delta+1}) + \epsilon)$. The specific running times are listed in Table 5.1 for graphs G with maximum degree Δ, where $3 \leq \Delta \leq 9$.

∎

Some comments about the results displayed in Table 5.1 are in place. First we will compare the running times for the algorithms that are dependent on the maximum degree of the input graph to the algorithm from Theorem 5.4, which running time is independent from the maximum degree of the input graph and furthermore provides the best worst-case time bound of any polynomial-space algorithm exactly solving 3-DNP as yet. The deterministic algorithm from Theorem 5.6 outperforms the general approach merely for the values $\Delta = 3$ and $\Delta = 4$. From this point of view, the probabilistic algorithm from Theorem 5.9 performs best, since it beats the general approach from Theorem 5.4 for the values Δ with $3 \leq \Delta \leq 8$, while the randomized approach from Theorem 5.7 has the overall lowest bound for two the cases $\Delta = 3$ and $\Delta = 4$.

Appendix A

Pseudo-Code of the Algorithm from Theorem 5.2

Algorithm for the Three Domatic Number Problem

Input: Graph $G = (V, E)$ with vertex set $V = \{v_1, v_2, \ldots, v_n\}$ and edge set E

Output: Partition of V into three dominating sets $D_1, D_2, D_3 \subseteq V$ or "failure"

if (*min-deg*$(G) \leq 1$) **output** "failure" and **halt**;
Set each of D_1, D_2, D_3, A_1, A_2, and A_3 to the empty set;
Set $R = V$;
Set $\mathcal{P} = (D_1, D_2, D_3, R)$;
Set $\mathcal{A} = (A_1, A_2, A_3)$;
DOMINATE$(G, \mathcal{P}, \mathcal{A})$; // Start recursion
output "failure" and **halt**;

Figure A.1: Algorithm for the Three Domatic Number Problem

```
Function DOMINATE(G, P, A) {          // P is a partition of graph G,
                                      // A is a triple of auxiliary sets
  RECALCULATE-GAPS(G, P, A);
  if (each D_i is a dominating set) {
    D_1 = D_1 ∪ R;
    output D_1, D_2, D_3;
  }
  if ( not HANDLE-CRITICAL-VERTEX(G, P, A)) {
    select vertex v ∈ R with
      deg(v) ≥ 3  and  maxgap_{P,A}(v) = maxgap_{P,A}(G) and
      sumgap_{P,A}(v) =
        max{sumgap_{P,A}(u) | u ∈ R ∧ maxgap_{P,A}(u) = maxgap_{P,A}(G)};
    find i with gap_{P,A}(v, i) = maxgap_{P,A}(v);
    ASSIGN(G, P, A, v, i);                          // First recursive call
    A_i = A_i ∪ {v};          // If recursion fails, put v in A_i and try again
    DOMINATE(G, P, A);                              // Second recursive call
  }
  return;
}
```

Figure A.2: Recursive function to dominate graph G

```
Function ASSIGN(G, P, A, v, i) {
  D_i = D_i ∪ {v};
  R = R − {v};
  DOMINATE(G, P, A);
}
```

Figure A.3: Function to assign vertex v to set D_i

Function RECALCULATE-GAPS$(G, \mathcal{P}, \mathcal{A})$ { // $\mathcal{P}$ is a partition of G,
// $\mathcal{A}$ is a triple of auxiliary sets

for all (vertices $v \in V$) {
if (vertex $v \in R$) {
for all $(i = 1, 2, 3)$ {
if $(v \notin A_i)$ {
$\text{gap}_{\mathcal{P},\mathcal{A}}(v, i) = ||N[v]|| - ||\{u \in N[v] \mid (\exists w \in N[u])\,[w \in D_i]\}||$;
} **else** { // $\perp$ indicates that $\text{gap}_{\mathcal{P},\mathcal{A}}(v, i)$ is undefined
$\text{gap}_{\mathcal{P},\mathcal{A}}(v, i) = \perp$;
}
}
$\text{maxgap}_{\mathcal{P},\mathcal{A}}(v) = \max_{i \in \{1,2,3\}}\{\text{gap}_{\mathcal{P},\mathcal{A}}(v, i)\}$;
$\text{mingap}_{\mathcal{P},\mathcal{A}}(v) = \min_{i \in \{1,2,3\}}\{\text{gap}_{\mathcal{P},\mathcal{A}}(v, i)\}$;
$\text{sumgap}_{\mathcal{P},\mathcal{A}}(v) = \sum_{i \in \{1,2,3\}} \text{gap}_{\mathcal{P},\mathcal{A}}(v, i)$;
}
$\text{openNeighbors}_{\mathcal{P}}(v) = \{u \in N[v] \mid u \in R\}$;
$\text{openSets}_{\mathcal{P}}(v) = \{i \in \{1, 2, 3\} \mid v \notin N[D_i]\}$;
$\text{balance}_{\mathcal{P}}(v) = ||\text{openNeighbors}_{\mathcal{P}}(v) - \text{openSets}_{\mathcal{P}}(v)||$;
}
$\text{maxgap}_{\mathcal{P},\mathcal{A}}(G) = \max_{v \in R}\{\text{maxgap}_{\mathcal{P},\mathcal{A}}(v)\}$;
$\text{mingap}_{\mathcal{P},\mathcal{A}}(G) = \min_{v \in R}\{\text{mingap}_{\mathcal{P},\mathcal{A}}(v)\}$;

}

Figure A.4: Function to recalculate gaps after partition has changed

```
Function boolean HANDLE-CRITICAL-VERTEX(G, P, A) {
  for all (vertices v ∈ V) {
    if (balance_P(v) < 0) {        // impossible to three dominate v with P
      return true;
    } else if (||{i ∈ {1,2,3} | v ∈ A_i}|| == 2) {
      select i with v ∉ A_i;                    // one choice for v remaining
      ASSIGN(G, P, A, v, i);
      return true;
    } else if (balance_P(v) == 0 and ||openSets_P(v)|| > 0) {
      select u ∈ N[v] ∩ R;                                   // v is critical
      for all (i with u ∉ A_i and v not dominated by D_i){
        ASSIGN(G, P, A, u, i);
      }
      return true;
    }
  }
  return false;                          // no critical vertices were found
}
```

Figure A.5: Function to handle the critical vertices

Bibliography

[AKS04] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. *Annals of Mathematics*, 160(2):781–793, 2004.

[BCGL92] S. Ben-David, B. Chor, O. Goldreich, and M. Luby. On the theory of average case complexity. *Journal of Computer and System Sciences*, 44(2):193–219, 1992.

[BE05] R. Beigel and D. Eppstein. 3-coloring in time $\mathcal{O}(1.3289^n)$. *Journal of the ACM*, 54(2):168–204, 2005.

[BH06] A. Björklund and T. Husfeldt. Inclusion-exclusion algorithms for counting set partitions. In *Proceedings of the 47th IEEE Symposium on Foundations of Computer Science*. IEEE Computer Society Press, 2006. To appear.

[Bon85] M. Bonuccelli. Dominating sets and dominating number of circular arc graphs. *Discrete Applied Mathematics*, 12:203–213, 1985.

[CGH+88] J. Cai, T. Gundermann, J. Hartmanis, L. Hemachandra, V. Sewelson, K. Wagner, and G. Wechsung. The boolean hierarchy I: Structural properties. *SIAM Journal on Computing*, 17(6):1232–1252, 1988.

[CGH+89] J. Cai, T. Gundermann, J. Hartmanis, L. Hemachandra, V. Sewelson, K. Wagner, and G. Wechsung. The boolean hierarchy II: Applications. *SIAM Journal on Computing*, 18(1):95–111, 1989.

[CH77] E. Cockayne and S. Hedetniemi. Towards a theory of domination in graphs. *Networks*, 7:247–261, 1977.

[CK96] R. Chang and J. Kadin. The boolean hierarchy and the polynomial hierarchy: A closer connection. *SIAM Journal on Computing*, 25(2):340–354, 1996.

[CM87] J. Cai and G. Meyer. Graph minimal uncolorability is D^P-complete. *SIAM Journal on Computing*, 16(2):259–277, 1987.

[Cob64] A. Cobham. The intrinsic computational difficulty of functions. In *Proceedings of the 1964 International Congress for Logic Methodology and Philosophy of Science*, pages 24–30. North Holland, 1964.

[Coo71] S. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd ACM Symposium on Theory of Computing*, pages 151–158. ACM Press, 1971.

[Edm65] J. Edmonds. Paths, trees and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[Epp03] D. Eppstein. The traveling salesman problem for cubic graphs. In *Proceedings of the 8th Workshop on Algorithms and Data Structures (WADS 2003)*, pages 307–318. Springer-Verlag *Lecture Notes in Computer Science #2748*, 2003.

[Epp04] D. Eppstein. Quasiconvex analysis of backtracking algorithms. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 788–797. ACM Press, 2004.

[EW00] W. Espelage and E. Wanke. Movement optimization in flow shop processing with buffers. *Mathematical Methods of Operations Research*, 51(3):495–513, 2000.

[Far84] M. Farber. Domination, independent domination, and duality in strongly chordal graphs. *Discrete Applied Mathematics*, 7:115–130, 1984.

[FGK05a] F. Fomin, F. Grandoni, and D. Kratsch. Measure and conquer: Domination—a case study. In *Proceedings of the 32th International Colloquium on Automata, Languages, and Programming*, pages 191–203. Springer-Verlag *Lecture Notes in Computer Science #3580*, 2005.

[FGK05b] F. 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.

[FGPS05] F. Fomin, F. Grandoni, A. Pyatkin, and A. Stepanov. Bounding the number of minimal dominating sets: A measure and conquer approach. In *Proceedings of the 16th International Symposium on Algorithms and Computation*, pages 573–582. Springer-Verlag *Lecture Notes in Computer Science #3827*, 2005.

[FHKS03] U. Feige, M. Halldórsson, G. Kortsarz, and A. Srinivasan. Approximating the domatic number. *SIAM Journal on Computing*, 32(1):172–195, 2003.

[Gil77] J. Gill. Computational complexity of probabilistic Turing machines. *SIAM Journal on Computing*, 6(4):675–695, 1977.

[GJ79] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[GJS76] M. Garey, D. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.

[GK00] V. Guruswami and S. Khanna. On the hardness of 4-coloring a 3-colorable graph. In *Proceedings of the 15th Annual IEEE Conference on Computational Complexity*, pages 188–197. IEEE Computer Society Press, 2000.

[GS87] Y. Gurevich and S. Shelah. Expected computation time for hamiltonian path problem. *SIAM Journal on Computing*, 16(3):486–502, 1987.

[Hem87] L. Hemachandra. The strong exponential hierarchy collapses. In *Proceedings of the 19th ACM Symposium on Theory of Computing*, pages 110–122. ACM Press, 1987.

[Hem89] L. Hemachandra. The strong exponential hierarchy collapses. *Journal of Computer and System Sciences*, 39(3):299–322, 1989.

[HHH98] E. Hemaspaandra, L. Hemaspaandra, and H. Hempel. What's up with downward collapse: Using the easy-hard technique to link boolean and polynomial hierarchy collapses. *SIGACT News*, 29(3):10–22, 1998.

[HHR97a] E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Exact analysis of Dodgson elections: Lewis Carroll's 1876 voting system is complete for parallel access to NP. *Journal of the ACM*, 44(6):806–825, 1997.

[HHR97b] E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Raising NP lower bounds to parallel NP lower bounds. *SIGACT News*, 28(2):2–13, 1997.

[HHS98a] T. Haynes, S. Hedetniemi, and P. Slater. *Domination in Graphs: Advanced Topics*. Marcel Dekker, 1998.

[HHS98b] T. Haynes, S. Hedetniemi, and P. Slater. *Fundamentals of Domination in Graphs*. Marcel Dekker, 1998.

[Hir00] E. Hirsch. New worst-case upper bounds for SAT. *Journal of Automated Reasoning*, 24(4):397–420, 2000.

[HMU01] J. Hopcroft, R. Motwani, and J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2nd edition, 2001.

[HS65] J. Hartmanis and R. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965.

[HT98] P. Heggernes and J. Telle. Partitioning graphs into generalized dominating sets. *Nordic Journal of Computing*, 5(2):128–142, 1998.

[HW97] E. Hemaspaandra and G. Wechsung. The minimization problem for boolean formulas. In *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science*, pages 575–584. IEEE Computer Society Press, 1997.

[HW02] E. Hemaspaandra and G. Wechsung. The minimization problem for boolean formulas. *SIAM Journal on Computing*, 31(6):1948–1958, 2002.

[JLL76] N. Jones, Y. Lien, and W. Laaser. New problems complete for nondeterministic log space. *Mathematical Systems Theory*, 10(1):1–17, 1976.

[Kad88] J. Kadin. The polynomial time hierarchy collapses if the boolean hierarchy collapses. *SIAM Journal on Computing*, 17(6):1263–1282, 1988. Erratum appears in the same journal, 20(2):404, 1991.

[Kar72] R. Karp. Reducibilities among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[KS94] H. Kaplan and R. Shamir. The domatic number problem on some perfect graph families. *Information Processing Letters*, 49(1):51–56, 1994.

[KSW87] J. Köbler, U. Schöning, and K. Wagner. The difference and truth-table hierarchies for NP. *R.A.I.R.O. Informatique théorique et Applications*, 21:419–435, 1987.

[Law76] E. Lawler. A note on the complexity of the chromatic number problem. *Information Processing Letters*, 5(3):66–67, 1976.

[Lev73] L. Levin. Universal sorting problems. *Problemy Peredaci Informacii*, 9:115–116, 1973. In Russian. English translation in *Problems of Information Transmission*, 9:265–266, 1973.

[Lev86] L. Levin. Average case complete problems. *SIAM Journal on Computing*, 15(1):285–286, 1986.

[Mil76] G. Miller. Riemann's hypothesis and tests for primality. *Journal of Computer and System Sciences*, 13:300–317, 1976.

[MM65] J. Moon and L. Moser. On cliques in graphs. *Israel Journal of Mathematics*, 3:23–28, 1965.

[MS72] A. Meyer and L. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *Proceedings of the 13th IEEE Symposium on Switching and Automata Theory*, pages 125–129. IEEE Computer Society Press, 1972.

[MS85] B. Monien and E. Speckenmeyer. Solving satisfiability in less than 2^n steps. *Discrete Applied Mathematics*, 10:287–295, 1985.

[Pap91] C. Papadimitriou. On selecting a satisfying truth assigment. In *Proceedings of the 32nd IEEE Symposium on Foundations of Computer Science*, pages 163–169, 1991.

[Pap94] C. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[PU59] M. Paull and S. Unger. Minimizing the number of states in incompletely specified state machines. *IRE Transactions on Electronic Computers*, EC-8:356–367, 1959.

[PW88] C. Papadimitriou and D. Wolfe. The complexity of facets resolved. *Journal of Computer and System Sciences*, 37(1):2–13, 1988.

[PY84] C. Papadimitriou and M. Yannakakis. The complexity of facets (and some facets of complexity). *Journal of Computer and System Sciences*, 28(2):244–259, 1984.

[PZ83] C. Papadimitriou and S. Zachos. Two remarks on the power of counting. In *Proceedings of the 6th GI Conference on Theoretical Computer Science*, pages 269–276. Springer-Verlag *Lecture Notes in Computer Science #145*, 1983.

[Rab80] M. Rabin. Probabilistic algorithms for testing primality. *Journal of Number Theory*, 12:128–138, 1980.

[Rob01] J. Robson. Finding a maximum independent set in time $\mathcal{O}(2^{n/4})$. Technical Report TR 1251-01, LaBRI, Université Bordeaux I, 2001.

[Rot03] J. Rothe. Exact complexity of Exact-Four-Colorability. *Information Processing Letters*, 87(1):7–12, 2003.

[Rot05] J. Rothe. *Complexity Theory and Cryptology. An Introduction to Cryptocomplexity*. EATCS Texts in Theoretical Computer Science. Springer-Verlag, 2005.

[RR04] T. Riege and J. Rothe. Complexity of the exact domatic number problem and of the exact conveyor flow shop problem. In *Proceedings of the First IEEE International Conference on Information & Communication Technologies: From Theory to Applications (ICTTA 2004)*, pages 653–654. IEEE Computer Society Press, 2004. A six-page extended abstract is available from CD-ROM ISBN 0-7803-8483-0.

[RR05] T. Riege and J. Rothe. An exact 2.9416^n algorithm for the three domatic number problem. In *Proceedings of the 30th International Symposium on Mathematical Foundations of Computer Science*, pages 733–744. Springer-Verlag *Lecture Notes in Computer Science #3618*, 2005.

[RR06a] T. Riege and J. Rothe. Completeness in the boolean hierarchy: Exact-Four-Colorability, minimal graph uncolorability, and exact domatic number problems—a survey. *Journal of Universal Computer Science*, 12(5):551–578, 2006.

[RR06b] T. Riege and J. Rothe. Complexity of the exact domatic number problem and of the exact conveyor flow shop problem. *Theory of Computing Systems*, 39(5):635–668, 2006.

[RR06c] T. Riege and J. Rothe. Improving deterministic and randomized exponential-time algorithms for the satisfiability, the colorability, and the domatic number problem. *Journal of Universal Computer Science*, 12(6):725–745, 2006.

[RRSY06a] T. Riege, J. Rothe, H. Spakowski, and M. Yamamoto. An improved exact algorithm for the domatic number problem. *Information Processing Letters*, 2006. To appear. Conference version: [RRSY06b].

[RRSY06b] T. Riege, J. Rothe, H. Spakowski, and M. Yamamoto. An improved exact algorithm for the domatic number problem. In *Proceedings of the Second IEEE International Conference on Information & Communication Technologies: From Theory to Applications (ICTTA 2006)*, pages 1021–1022. IEEE Computer Society Press, 2006.

[RSV03] J. Rothe, H. Spakowski, and J. Vogel. Exact complexity of the winner problem for Young elections. *Theory of Computing Systems*, 36(4):375–386, 2003.

[RW01] S. Reith and K. Wagner. On boolean lowness and boolean highness. *Theoretical Computer Science*, 261(2):305–321, 2001.

[Sch78] T. Schaefer. The complexity of satisfiability problems. In *Proceedings of the 10th ACM Symposium on Theory of Computing*, pages 216–226. ACM Press, 1978.

[Sch99] U. Schöning. A probabilistic algorithm for k-SAT and constraint satisfaction problems. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science*. IEEE Computer Society Press, 1999.

[Sch05] U. Schöning. Algorithmics in exponential time. In *Proceedings of the 22th Annual Symposium on Theoretical Aspects of Computer Science*, pages 36–43. Springer-Verlag *Lecture Notes in Computer Science #3404*, 2005.

[SG76] S. Sahni and T. Gonzalez. P-complete approximation problems. *Journal of the ACM*, 23(3):555–565, 1976.

[SS77] R. Solovay and V. Strassen. A fast Monte Carlo test for primality. *SIAM Journal on Computing*, 6:84–85, 1977. Erratum appears in the same journal, 7(1):118, 1978.

[Ste90] R. Stearns. Juris Hartmanis: The beginnings of computational complexity. In A. Selman, editor, *Complexity Theory Retrospective*, pages 1–18. Springer-Verlag, 1990.

[SU02a] M. Schaefer and C. Umans. Completeness in the polynomial-time hierarchy: Part I: A compendium. *SIGACT News*, 33(3):32–49, 2002.

[SU02b] M. Schaefer and C. Umans. Completeness in the polynomial-time hierarchy: Part II. *SIGACT News*, 33(4):22–36, 2002.

[Tur36] A. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society, ser. 2*, 42:230–265, 1936. Correction, *ibid*, vol. 43, pp. 544–546, 1937.

[Uma98] C. Umans. The minimum equivalent DNF problem and shortest implicants. In *Proceedings of the 39th IEEE Symposium on Foundations of Computer Science*, pages 556–563. IEEE Computer Society Press, 1998.

[Wag87] K. Wagner. More complicated questions about maxima and minima, and some closures of NP. *Theory of Computing Systems*, 51:53–80, 1987.

[Wag90] K. Wagner. Bounded query classes. *SIAM Journal on Computing*, 19(5):833–846, 1990.

[Wan97] J. Wang. Average-case computational complexity theory. In L. Hemaspaandra and A. Selman, editors, *Complexity Theory Retrospective II*, pages 295–328. Springer-Verlag, 1997.

[Wil84] H. Wilf. Backtrack: An $\mathcal{O}(1)$ expected time algorithm for the graph coloring problem. *Information Processing Letters*, 18(3):119–121, 1984.

[Woe03] G. Woeginger. Exact algorithms for NP-hard problems: A survey. In *Combinatorial optimization - eureka, you shrink!*, pages 185–207. Springer-Verlag *Lecture Notes in Computer Science #2570*, 2003.

[Yam05] M. Yamamoto. An improved $\mathcal{O}(1.234^m)$-time deterministic algorithm for SAT. In *Proceedings of the 16th International Symposium on Algorithms and Computation*, pages 644–653. Springer-Verlag *Lecture Notes in Computer Science #3827*, 2005.