On the Complexity of Alternative Solutions

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

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

von Diplom-Mathematiker Michael Krüger
geboren am 9. Januar 1980 in Jena

Gutachter

1. PD Dr. Harald Hempel, Friedrich-Schiller-Universität Jena
2. Prof. Dr. Jörg Rothe, Heinrich-Heine-Universität Düsseldorf

Tag der letzten Prüfung des Rigorosums: 12. 07. 2008
Tag der öffentlichen Verteidigung: 23. 07. 2008

Zusammenfassung

Diese Dissertation untersucht die Komplexität alternativer Lösungen. Das heißt, wir betrachten die Frage, ob eine oder mehrere gegebene Lösungen eines Problems, das Finden weiterer Lösungen vereinfacht. In der Praxis relevant ist diese Fragestellung zum Beispiel wenn sich eine mit großem Aufwand berechnete Lösung eines schwierigen Problems im Nachhinein als unzureichend erweist. In diesem Falle ist es notwendig nach alternativen Lösungen zu suchen, wobei nun die bereits gefundene Lösung als Ausgangspunkt der Berechnung genutzt werden kann. Darüber hinaus hat die untersuchte Aufgabenstellung eine Bedeutung in der Erstellung von (auch hier immer beliebteren) japanischen Rätseln wie Sudoku, Kakkuro oder Nurikabe. Beispielsweise werden im Fall von Sudoku, ausgehend von einem vollständig ausgefüllten Gitter (Startlösung), Ziffern so gestrichen dass die Startlösung die eindeutige Lösung des Rätsels bleibt. Dazu muss während des Streichprozesses wiederholt geprüft werden, ob es neben der Startlösung alternative Lösungen gibt.

Im ersten Teil der Arbeit (Kapitel 3 und 4) betrachten wir die Klasse der NP-vollständigen Probleme. Wir formalisieren den Begriff der Lösung mittels sogenannter Verifier und das Problem alternativer Lösungen für NP-Sprachen. Indem wir die Härte des Problems alternativer Lösungen für einige Probleme zeigen, motivieren wir die Vermutung, dass eine gegebene Lösung das Finden alternativer Lösungen nicht vereinfacht. Wir entwickeln den Begriff des *universellen Verifiers*, der es ermöglicht, einen geeigneten Lösungsbegriff für ein Problem formal zu charakterisieren. Darüber hinaus zeigen wir, dass es möglich ist, mit einer einzigen sogenannten $\exists_r \forall_l gp$-Reduzierung einen Lösungsbegriff für ein Problem als geeignet zu identifizieren sowie die Härte des Problems alternativer Lösungen für jede Anzahl gegebener Lösungen zu zeigen. Unter Benutzung dieser Reduzierung, erhärten wir die obige Vermutung, indem wir für eine große Zahl NP-vollständiger Probleme wie zum Beispiel 0/1-Integer Programming, 3Dimensional Matching, Minimum Edge Cost Flow und Vertex Cover zeigen, dass bezüglich eines geeigneten Lösungsbegriffes alternative Lösungen nicht leicht zu berechnen sind.

Darüber hinaus übertragen wir die Theorie für NP-Probleme auch auf die Klasse RE der aufzählbaren Sprachen (Kapitel 5) und die Klassen Σ_i^p der Polynomialzeithierarchie (Kapitel 6). Für RE zeigen wir damit, dass das Problem alternativer Lösungen für RE wenig sinnvoll ist, da wir für jedes RE-Problem einen geeigneten Lösungsbegriff finden, der höchstens eine Lösung zulässt. Die Situation in der Polynomialzeithierarchie ist vermutlich ähnlich zum NP-Fall. Für Σ_2^p können wir

die Härte des Problems alternativer Lösungen für einige typische Probleme zeigen, z.B. für Generalized Subset Sum und Strongest Implicant Core. Deshalb und wegen der starken strukturellen Ähnlichkeit zu NP (Die Polynomialzeithierarchie ist eine Verallgemeinerung von NP, insbesondere gilt $\Sigma_1^p = \mathrm{NP}$) vermuten wir auch hier, dass alternative Lösungen im Allgemeinen genauso schwer zu finden sind, wie eine erste Lösung.

Der Hauptteil der Arbeit beschäftigt sich also mit Entscheidungsproblemen aus NP, RE und der Polynomialzeithierarchie und kommt zu dem Ergebnis, dass alternative Lösungen schwer zu berechnen sind. In Kapitel 7 untersuchen wir deshalb die Frage, ob eine gegebene Lösung möglicherweise dabei hifreich ist, eine alternative Lösung näherungsweise zu berechnen, also zu approximieren. Es zeigt sich, dass es im Falle der Approximierbarkeit durchaus Probleme gibt, bei denen eine alternative Lösung leichter zu approximieren ist. Neben der Betrachtung einiger Probleme, bei denen das trivialerweise der Fall ist (eine minimale Änderung der gegebenen optimalen Lösung liefert stets eine gute Approximation) geben wir einige gehaltvolle positive Resultate an, zum Beispiel für MinMaxMatching und Minimum Steiner Tree. Im Gegensatz dazu, zeigen wir für Probleme wie Minimum Independent Dominating Set und Minimum Traveling Salesperson Problem, dass alternative Lösungen nicht besser approximierbar sind, als eine erste Lösung.

Im letzten Kapitel (Kapitel 8) beantworten wir eine offene Frage aus einem verwandten Gebiet, nämlich dem Gebiet der inversen Probleme. Das inverse Problem zu einem NP-Problem besteht darin, für eine gegebene Menge von Lösungen zu entscheiden, ob es eine Probleminstanz mit genau den gegebenen Lösungen gibt. Dazu kann man oft zunächst einen Kandidaten berechnen, der mindestens die gegebenen Lösungen hat und diskutiert anschließend, ob dieser Kandidat alternative Lösungen hat. Hat der Kandidat keine alternativen Lösungen, so ist er solch eine gesucht Probleminstanz, und falls der Kandidat alternative Lösungen besitzt, dann gibt es keine solche Instanz. Wir beantworten die Frage nach der Komplexität des inversen Problems von 3Dimensional Matching, indem wir seine coNP-Vollständigkeit zeigen. Damit ist die coNP-Vollständigkeit auch für das letzte der sechs NP-Basisprobleme (von Garey und Johnson) 3Satisfiability, 3Dimensional Matching, Vertex Cover, Clique, Hamiltonian Cycle und Partition gezeigt. Darüber hinaus untersuchen wir noch die Verträglichkeit unseres Begriffes für geeignete Lösungen mit inversen Problemen und geben starke Argumente für eine gewisse Unverträglichkeit an.

Acknowledgements

I would like to thank everybody who supported me during my work on this thesis. First of all I want to thank my supervisor Harald Hempel for suggesting to work in this field and the excellent guidance during my time as PhD student and the work on my diploma thesis. He was a steady source for advice and inspiration, but also for encouragement.

Special thanks to my colleague and friend Tobias Berg, who had a helping hand, ear, or brain whenever it was needed. By helping to paper my new flat, being a challenging jogging partner and much more he was (and is) a great support outside the university. His contribution to this work is surely even higher valued. The door to his office was open whenever something had to be discussed, examined, explained or proofread, which was an enormous help especially in tough times.

Furthermore, my thank goes to all members of our "lunch group". The interesting, entertaining, or enlightening lunch talks with Harald, Tobi, Marcus, Dave, Karsten, Madlen, Christian, Ronny, and all the friends I might have forgotten here, were a welcome change from staring at the monitor.

Many thanks also go to Mick who abused his universities online-access to almost all journals to provide me with a lot of needed papers.

I also want to mention all the people who shared the office with me. Thanks for the good time and the useful hints about this and that go to Hannes, Somnath, Nadja, Matthias, and Thomas.

Last, but not least, I am very grateful to all of my friends and family members. Without their steady support, this thesis would have been impossible.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation and Overview

Computational theory has developed several concepts to deal with NP-complete problems, e.g., fast exponential-time exact algorithms, fixed-parameter algorithms, approximation algorithms, and heuristics. To get an idea of this thesis' contribution to the field of coping with NP-completeness, consider the following scenario that actually happened to a colleague.

Imagine you are working in automotive supply industry. You are given a large network representing the tree of cables in a car. The network contains two types of vertices, namely control unit vertices and route vertices. Using an edge (cable) e *causes the costs* c(e)*. Your task is to find a tree that connects all control units and whose edge cost sum is below a given limit. As a skilled computer scientist you quickly realize that the given problem equals* Steiner Tree*, which is an* NP*-complete problem. You decide to implement an exponential-time algorithm and after several days of computation you proudly present the solution the algorithm has found. But the engineer rejects the tree you found because of technical reasons and without further explanations asks for another one, just saying "Having a valid tree, it should be easy to generate further ones".*

The topic of this thesis is the question if the engineer was right, that is, does having a solution of a given instance help in finding alternative solutions.

After stating some notions and giving some basic definitions in Chapter 2, we start studying this question for decision problems from NP (Chapter 3). We formalize the notion of solution using *verifiers* and define the alternative solution problem $n + A$ of a problem A for each natural number as follows.

- **Given:** An instance x with n different solutions $y_1, \ldots, y_n$.
- **Question:** Is there another solution y_{n+1} for x ?

In order to do so we concentrate on the verifier (notion of solutions) that appears to be the most natural one. Using a rather naive approach, we gather some first

results, which are all negative, that is, all alternative solution problems $n + A$ are NP-complete for NP-complete problems A like e.g. KERNEL and SUBSET SUM. In order to achieve these results we give a many one reduction $A \leq_m^p n + A$, for each natural number n. Since hardness of $n + A$ is derived from its basis problem A we call this kind of reduction an *auto*-reduction.

Using another type of reduction we can achieve the same result by giving only one so-called *generating parsimonious reduction* (*gp*-reduction) from A to $1 + A$, which is still an auto-reduction. Using the generating parsimonious reduction one can also transfer hardness results from one problem to another. In particular, having the NP-completeness for all $n + A$, $n \in \mathbb{N}$, one *gp*-reduction from A to B is sufficient to gain the NP-completeness for all $n + B$, $n \in \mathbb{N}$. We call this kind of reduction an *inter*-reduction.

In Chapter 4, instead of immediately applying these concepts, we first concentrate on another interesting issue, namely the question for the proper notion of solutions. Formally solutions are characterized via verifiers, but there exists an infinite number of verifiers for each NP-problem. So, which is the right one? Can we formally justify the use of the natural verifiers? We provide an answer to this question, by introducing the notion of universal verifiers. A verifier V_A for a problem A is called universal if all verifiers for A can be *gp*-reduced to V_A. Based on *gp*-reductions we develop the $\exists_r \forall_l$*gp-reduction* which allows for showing the NP-completeness of all alternative solution problems $n + A$ and for identifying the used verifier as proper (universal). Using these concepts we show that the natural verifier is universal and that the alternative solution problem is NP-complete for a lot of languages, e.g., 0/1-INTEGER PROGRAMMING, 3DIMENSIONAL MATCHING MINIMUM EDGE COST FLOW and VERTEX COVER. These results motivate the conjecture that the alternative solution problems defined with respect to natural verifiers are hard for all NP-problems and that all natural verifiers are also universal. A counterexample to this conjecture seems to be the problem if finding Hamiltonian cycles in cubic graphs with its natural verifier. We discuss this especial problem and try to find a way out of this seeming contradiction by finding another natural and universal verifier.

In Chapter 5, we transfer these concepts from NP to RE. Without problems, we can define the respective notions for RE. We also manage to show that all alternative solution versions of a special version of the halting problem are RE-complete and that its natural verifier is universal. But in case of the standard halting problem it turns out that the natural verifier is not universal. Instead, another unnatural verifier is universal for the standard halting problem, which collides with our understanding of universal verifiers as proper verifiers. We find a reason for that in the property, that each RE-problem has a quite natural verifier such that for each instance exists at most one solution. Eying such verifiers discussing alternative solutions in RE is more or less senseless.

In Chapter 6 we use our theory to discuss the classes Σ_i^p from the polynomial-time hierarchy. Again, we can easily transfer all notions and it turns out that the situation in Σ_i^p is close to the case of NP. We show the Σ_i^p-completeness

for all alternative solution problems of Σ_i^p-SAT and find out that the associated natural verifiers are universal. In case of Σ_2^p we furthermore obtain the same result for several more languages, like Generalized Subset Sum and Strongest Implicant Core. We conjecture that the classes Σ_i^p behave similar to NP, that is, all alternative solution problems are hard and the natural verifiers are universal.

After focusing on decision problems and obtaining mostly negative results in the Chapters 3, 4, 5, and 6, in Chapter 7 we also consider the approximability of alternative solution for some optimization versions of NP-problems. Here, a given optimal solution often helps in approximating the second best solution, more precisely, approximating the optimal solution without hints is often substantially harder than approximating the second best solution knowing the optimal solution, for instance, for Minimum Steiner Tree and MinMaxMatching. We also give examples for problems for which approximating the second best solution (using the optimal solution) is as hard as approximating the optimal solution (without hints), namely Minimum Traveling Salesperson Problem and Minimum Independent Dominating Set.

Finally, in Chapter 8 we give an interesting result in the field of inverse problems. The inverse problem is, given a set of solutions, to decide whether there exists an instance having exactly those solutions. We show the coNP-completeness for the inverse problem of 3Dimensional Matching which completes the coNP-completeness results [KS99, Che03, Krü05] for the six basic NP-complete problems from [GJ79]. Using Satisfiability as an example, we also give strong arguments for the fact, that two different universal verifiers for the same problem sometimes induce inverse problems of different complexity[1].

1.2 Previous work

The problem of finding alternative solutions was firstly stated and studied by Ueda and Nagao in the context of puzzles like, e.g., Sudoku, Kakkuro, and Nonogram [UN96]. The question for alternative solutions is of crucial interest for designing such puzzles, because they are supposed to have unique solutions. In several papers, the group of Ueda and Nagao showed the NP-completeness of the puzzles, Slither Link, Sudoku, Fillomino, Kakkuro, Nonogram, and their alternative solution versions [UN96, Set02, YS02]. As a byproduct, they also showed the hardness of the alternative solution problems of Satisfiability, 3Satisfiability, 3Dimensional Matching and Hamiltonian Cycle. Their concepts were also used to show the NP-completeness of Cryptarisms, Minesweeper, and the respective alternative solution problems, in [McP03, dB04]. Unfortunately, we discovered the mentioned results after we independently developed some of the above concepts and results.

[1] In contrast, the alternative solution problems defined with respect to two different universal verifiers are equally hard.

Some singular results about approximating the second best solution using the known optimal solution can be found in [BST99] and [AH98]. In [BST99], the authors give a polynomial-time approximation scheme for approximating the second best longest cycle in a Hamiltonian cubic graph, and in [AH98], it is shown that the second best version of MAP (finding maximum a posteriori assignments), a problem in the field of artificial intelligence, admits no constant approximation, unless P = NP.

Another approach to our introductory scenario for NP-problems was studied in, e.g., [SM93, vdPLSvdV99]. Instead of exhaustively computing one solution and, afterwards, deriving alternative solutions from this optimal solution (which is our approach), it is tried to efficiently compute the k best solutions in the first exhaustive step and thus, to provide alternative solutions, a priori. In particular, in [SM93] a fast exponential-time algorithm computing the k best solutions of Chinese Postman is given and in [vdPLSvdV99] the authors give an algorithm for the k-best Minimum Traveling Salesperson Problem. The k-best version has also been studied for some P-problems, e.g., Minimum Cut and Perfect Matching [HQ85].

Chapter 2

Preliminaries

In this chapter we define the basic concepts and notions that are used in this thesis. Almost all of them can be found in the standard books on computational complexity, for instance [BDG90, BDG88, Pap94, WW86]. The graph-theoretical parts can be found in [Wes96]. We assume that the reader is familiar with the basic set theoretic and logical concepts and notations.

2.1 Words and languages

Let $\mathbb{N}$ denote the set of natural numbers and $\mathbb{N}^+ = \mathbb{N} \setminus \{0\}$ the set of positive natural numbers.

In complexity theory, problems (languages) are sets of words over a finite alphabet. Unless otherwise noted, we always use $\Sigma = \{0, 1\}$ as our standard alphabet. The empty word is denoted by ϵ. The symbol Σ^n denotes the set of all words consisting of n letters from Σ. The set $\Sigma^* = \bigcup_{i \in \mathbb{N}} \Sigma^n$ contains all finite words over Σ. The length of a word x is denoted by $|x|$. The complement of a language $A \subseteq \Sigma^*$ is denoted by $\overline{A}$.

Let $\leq_{lex}$ denote the quasi lexicographical ordering on Σ^*. For words u and v, $u \leq_{lex} v$ if and only if either $u = v$, or $|u| \leq |v|$, or $|u| = |v|$ and there exist some $w, u', v' \in \Sigma^*$ such that $u = w0u'$ and $v = w1v'$.

The elements of the treated problems are often graphs, formulas, sets, functions or other structures, but not words. To fit in the above scheme we use easily computable encodings of graphs, formulas, sets, functions, etc. to strings from Σ^* such that the size of the encoding properly reflects the size of the encoded object. It is not hard to show the existence of such encodings. To avoid formalism we speak of graphs, formulas, sets, functions etc. as members of problems instead of their encodings as words.

We often deal with languages A of pairs or tuples of words, which do also not fit into the scheme of languages as subsets of Σ^*. In this case, we use pairing functions σ_k from $(\Sigma^*)^k$ to Σ^*, $k \in \mathbb{N}$, which are polynomial-time computable and polynomial-time invertible. To focus on the essential parts of argumentations, we just write $(x_1, \ldots, x_k) \in A$, when we mean $\sigma_k((x_1, \ldots, x_k)) \in A$.

2.2 Turing machines

Our computational model is the concept of multi-tape Turing machines. We refer to [WW86] for a formal definition of Turing machines. We just mention the following conventions. A Turing machine M has two halting states, an accepting and a rejecting halting state, and we say that a Turing machine halts if and only if it reaches one of its halting states. We say that a Turing machine accepts (rejects) an input x if and only if the computation of M on the input x reaches the accepting (rejecting) halting state.

Let f be a mapping from $\mathbb{N}$ to $\mathbb{N}$. Then we say that a Turing machine works in time f if and only if it halts after at most $f(|x|)$ computational steps, for all inputs $x \in \Sigma^*$. Following this, a Turing machine works in polynomial-time if and only if it works in time p for some polynomial p. When dealing with running times of algorithms we assume, without loss of generality, that polynomials are monotonically increasing.

We will use deterministic and nondeterministic Turing machines. The computation of a nondeterministic or deterministic Turing machine M on the input x is denoted by $M(x)$. A deterministic Turing machine (DTM) M accepts a language $A \subseteq \Sigma^*$ if and only if, for all $x \in \Sigma^*$, $M(x)$ accepts if $x \in A$ and $M(x)$ rejects if $x \notin A$. A nondeterministic Turing machine accepts a language $A \subseteq \Sigma^*$ if and only if, for all inputs $x \in \Sigma^*$, there is an accepting path in the computation of $M(x)$ if and only if $x \in A$.

The program of a Turing machine is a finite text of a certain structure over a finite alphabet. So, the set $\mathcal{P}$ of programs for Turing machines is countable and there is a bijection $g : \mathcal{P} \to \mathbb{N}$. It is not hard to see, that there also exist such bijections that are computable and invertible in polynomial-time, so-called Gödel numberings for Turing machines. For this thesis, we fix one Gödel numbering and denote the machine with the Gödel number i with M_i.

A Turing machine M can be equipped with an oracle $A \subseteq \Sigma^*$. Such an oracle Turing machine M^A has a special query tape in order to test membership of words in A. When the machine reaches a special query state it receives the answer "Yes" if the word on the query tape is in A and "No", otherwise. This answer requires only one computational step.

2.3 Complexity classes and reductions

A complexity class is a set of problems with the same complexity in terms of resources of a computational model, for instance Turing machines. The class P is the set of problems that can be accepted by a DTM in polynomial time. Similarly, NP contains the problems that are accepted by a nondeterministic polynomial-time Turing machine. The class UP is the set of all problems that can be accepted by a nondeterministic polynomial-time Turing machine, whose computation, for all inputs $x \in \Sigma^*$, has at most one accepting computational path.

The class REC contains exactly those problems that can be accepted by a DTM in finite time. The class RE is the collection of problems A for which there exists a DTM M such that the computation of M on input x halts and accepts after a finite number of steps if $x \in A$, but on the other hand, if $x \notin A$ the machine does not halt.

A DTM M with an output tape computes a function $g : \Sigma^* \to \Sigma^*$ if and only if the computation $M(x)$ halts with the string $g(x)$ on the output tape, for all $x \in \Sigma^*$. A function g, computable by some Turing machine, is called recursive.

A DTM M computes a function $g : \Sigma^* \to \Sigma^*$ in time $f : \mathbb{N} \to \mathbb{N}$ if and only M computes g and halts after at most $f(|x|)$ computational steps, for all $x \in \Sigma^*$. The class FP contains all functions that are computable by a DTM in time p, for some polynomial p. A function is computable in linear-time if it can be computed by a DTM in time $c \cdot n$, for some natural number c.

A problem A is called many-one reducible to a problem B ($A \leq_m B$) if and only if there exists a total recursive function f, such that $x \in A \leftrightarrow f(x) \in B$, for all $x \in \Sigma^*$. A many-one reduction via f is called a polynomial-time many-one reduction ($\leq_m^p$) if and only if $f \in \mathrm{FP}$. When dealing with the classes P and NP we always use the polynomial-time many-one reduction. For simplification, in this case, we just use the term many-one reduction. A problem A is called complete for a complexity class $\mathcal{C}$ via a type of reduction $\leq$ if and only if $A \in \mathcal{C}$ and $B \leq A$, for all $B \in \mathcal{C}$.

2.4 Graph theory

An undirected graph G is an ordered pair (V, E), where V (vertices) is a finite set and E (edges) is a subset of $\{\{u, v\} : u \neq v \wedge u, v \in V\}$. If $\{u, v\} \in E$, we say that u and v are connected by an edge. The number of edges, a vertex v is contained in, is called degree of v and denoted by $d_G(v)$. A directed G graph is an ordered pair (V, A), where V (vertices) is a finite set and A (arcs) is a subset of $V \times V \setminus \{(u, u) : u \in V\}$. An element (u, v) of A is called an arc from u to v. To distinguish between directed and undirected graphs, we call directed graphs digraphs and mean undirected graphs, when we just say graph.

An (un)directed path from u to v in a directed (undirected) graph G is a sequence of vertices $u = w_0, w_1, \ldots, w_{k-1}, w_k = v$ such that there is an arc (edge) from w_i to w_{i+1} in G, for all $0 \leq i \leq k - 1$. A simple path is a path, where $w_i \neq w_j$, for all $i \neq j$. A cycle is a path from u to v, where $u = v$ and $w_i \neq w_j$, for all i and j with $i \neq j$ and $\{i, j\} \neq \{0, k\}$. A cycle that traverses all vertices of a graph G is called a Hamiltonian cycle of G. Sometimes, we identify a cycle with the set of participating edges or arcs.

For a given undirected graph $G = (V, E)$ and a subset $V' \subseteq V$ of vertices, we defined the subgraph $G[V']$ of G that is induced by V' as $G[V'] = (V', E')$, where $E' = \{\{u, v\} : \{u, v\} \in A \text{ and } u, v \in V'\}$. Analogous, for a directed graph $G = (V, A)$, the subgraph graph $G[V']$ induced by a set $V' \subseteq V$ of vertices is

$G[V'] = (V', A')$, where $A' = \{(u, v) : (u, v) \in A \text{ and } u, v \in V'\}$.

The graph consisting of n vertices and all possible edges between these vertices is called the complete graph with n vertices and denoted by K_n. The graph that is just a simple cycle with n vertices is referred to as C_n. For a graph $G = (V, E)$ the term $\overline{G}$ denotes the complement of G, that is $\overline{G} = (V, \overline{E})$ with $\overline{E} = \{\{u, v\} : \{u, v\} \subseteq V \wedge u \neq v\} \setminus E$.

2.5 Propositional logic

The literals of a variable x are x and $\neg x$. A clause C over a variable set X is a formula $C = \ell_1 \vee \ell_2 \vee \cdots \vee \ell_n$, $n \in \mathbb{N}$, where ℓ_j is a literal of a variable from X, for all $1 \leq j \leq n$. A Boolean formula F over the variable set X is in conjunctive normal form (CNF) if $F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$, $m \in \mathbb{N}$, where C_i is a clause over X, for all $1 \leq i \leq m$. A Boolean formula in CNF is called a 3CNF-formula if and only if all clauses contain exactly three literals.

A monomial T over a variable set X is a formula $T = \ell_1 \wedge \ell_2 \wedge \cdots \wedge \ell_n$, $n \in \mathbb{N}$, where ℓ_j is a literal of a variable from X, for all $1 \leq j \leq n$. A Boolean formula F over the variable set X is in disjunctive normal form (DNF) if $F = T_1 \vee T_2 \vee \cdots \vee T_m$, $m \in \mathbb{N}$, where T_i is a monomial over X, for all $1 \leq i \leq m$. Monomials and clauses over a variable set X are often represented as sets of literals of variables from X. A Boolean formula in DNF is called a 3DNF-formula if and only if all monomials contain exactly three literals.

Chapter 3

Alternative Solutions in NP I - Concepts to Show Hardness

The objective of this thesis is to examine the complexity of the problem of alternative solution, which is to decide, given an instance and several solutions, if there are further solutions for the given instance. In this chapter we settle the very basics by formalizing the notion of the alternative solution problems (ASP's) for an NP-problem. Given an NP-problems A, we define an ASP $n + A$, for each natural number n of given solutions.

We present some concepts to show the hardness of all alternative solution problems $n + A$. This can be done by reducing the NP-problem A to its ASP's $n + A$, so called auto-reductions. Note that to do so, an infinite number of reductions is needed. Moreover, we introduce a type of reduction, by which this result can be achieved with a single reduction, the generating parsimonious reduction. The generating parsimonious reduction also allows for comparing the complexity of the ASP's of different NP-problems, the so called inter-reduction. We also show that for all so-called connected subgraph problems A, all alternative solution problems $n + A$ are hard. Note that some of the introduced notions and results have been found independently in [UN96, Set02, YS02], where the focus is on alternative solutions of puzzles and games.

For each of the presented concepts to show the hardness of ASP's, we give some examples, ending up in a list of problems with NP-complete ASP's suggesting that the ASP's of NP-complete problems might be hard in general.

3.1 Preliminaries

In this section we formally define the notion of solutions for NP-problems, using the concept of NP-verifiers. Then, we formally introduce the problems of alternative solutions for NP-problems and state some simple properties.

Obviously, the first step in studying the complexity of alternative solutions of NP-complete problems is to formalize the notion of a solution. To do so we use

the following well-known characterization of the class NP.

Theorem 3.1.1. *A language A is in* NP *if and only if there exist a language B from* P *and a polynomial p such that for all $x \in \Sigma^*$, $x \in A \leftrightarrow (\exists y)\ [|y| \leq p(|x|) \wedge (x, y) \in B]$.*

So, an NP-computation can be imagined as a two phased process. The existential quantifier represents the first phase, nondeterministically guessing a polynomial length bounded solution y for x. In the second (deterministic) phase the P-predicate B verifies if y actually is a solution for x.

Languages from P that can play the role of the language B in Theorem 3.1.1 are called polynomial-time verifiers.

Definition 3.1.2. *1. A polynomial-time acceptable language V is called a* polynomial-time verifier (or NP-verifier) *if and only if there exists a polynomial p satisfying, $(\forall x, y \in \Sigma^*)[(x, y) \in V \rightarrow |y| \leq p(|x|)]$.*

2. The language $L(V)$ accepted by a polynomial-time verifier V is defined as $L(V) = \{x \in \Sigma^* : (\exists y \in \Sigma^*)[(x, y) \in V)]\}$.

Throughout this thesis we will use the term verifier as a shorthand for polynomial-time verifier.

Now, we can introduce the term of a solution for a language A from NP with respect to a verifier V_A for A.

Definition 3.1.3. *Let A be a language from* NP *and let V_A be a verifier such that $L(V_A) = A$. We call a string y a* solution *for an instance x of A (with respect to V_A), if $(x, y) \in V_A$. The set of solutions for an instance x is denoted with $V_A(x) = \{y \in \Sigma^* : (x, y) \in V_A\}$.*

Note that the concept of solutions for instances of an NP-language A depends on the chosen verifier V_A. The common descriptions of NP-problems A often fit the scheme that is given in Theorem 3.1.1. For instance, SAT is the set of CNF-formulas F such that there exists an assignment α with $F(\alpha) = 1$. The choice of the verifier $V_{\text{SAT}} = \{(F, \alpha) : F \text{ is a CNF-formula and } F(\alpha) = 1\}$ for SAT seems to be very natural.

First, when studying solutions of NP-problems, we will examine solutions with respect to those verifiers that we feel are the most natural ones.

Now, we are able to define the alternative solution problem for an NP-problem with an associated verifier.

Definition 3.1.4. *Let A be a problem in* NP *with a verifier V_A and let $n \geq 1$ be a natural number. The alternative solution problem (ASP) $n + A_{V_A}$ for A is the following:*

Given: *A tuple $(x, y_1, \ldots, y_n)$ such that all y_i are pairwise different solutions for x, i.e., for all $1 \leq i \leq n$, $(x, y_i) \in V_A$ and for all $1 \leq i < j \leq n$, $y_i \neq y_j$.*

Question: *Is there another solution y_{n+1} for x, i.e., is there a string y_{n+1} such that $(x, y_{n+1}) \in V_A$ and for all $1 \leq i \leq n$, $y_i \neq y_{n+1}$?*

By formal means we furthermore define $0 + A_{V_A}$ as A.

As mentioned above, when it comes to discussing concrete problems we will determine the most natural verifier and discuss solutions with respect to this verifier. After fixing the natural verifier V_A for a problem A we will write $n + A$ as a shorthand for $n + A_{V_A}$.

In the remainder of this section we will state some properties of the operation "+". The next theorem gives a (quite obvious) upper bound for the complexity of $n + A$ for any natural number n and NP-problem A.

Theorem 3.1.5. *Let A be a problem from* NP*, let V_A be a verifier for A, and let n be a natural number. Then $n + A$ (defined with respect to V_A) is also in* NP*.*

Proof. Consider the following predicate V_{n+A}:

$$V_{n+A} = \{((x, y_1, \ldots, y_n), y) : (\forall\ 1 \leq i \leq n)[(x, y_i) \in V_A], (\forall\ 1 \leq i \neq j \leq n)[y_i \neq y_j], (\forall\ 1 \leq i \leq n)[y_i \neq y] \wedge (x, y) \in V_A\}.$$

So, V_{n+A} accepts a strings y as a solution for an instance $(x, y_1, \ldots, y_n)$ with $(x, y_i) \in V_A$, $1 \leq i \leq n$, if and only if y is a solution for x ($(x, y) \in V_A$) different from $y_1, \ldots, y_n$. It is not hard to see that V_{n+A} is a polynomial-time verifier for $n + A$. Hence, by Theorem 3.1.1 $n + A$ is in NP. □

Note that the given verifier V_{n+A} for $n+A$ arises from V_A in a very canonical way. So in the following, for a given problem A and an associated verifier V_A we will use the above defined verifier V_{n+A} as standard verifier for $n+A$. Later we will develop the concept of universal verifiers, which formally characterizes "proper"verifiers (see Chapter 4). For all treated problems A we will then see that if V_A is a proper (universal) verifier for A, then the verifier V_{n+A} is also proper (universal) for $n+A$, which strongly legitimates the use of V_{n+A} as standard verifier for $n + A$.

The following theorem provides an associativity-like property of "+".

Theorem 3.1.6. *Let A be a language from* NP *with an associated verifier V_A. Then for each pair of natural numbers n and m, $n + (m + A)$, and $(n + m) + A$ are many-one equivalent via linear-time functions.*

Proof. The proof can be done by formally applying the definition of $+nm + A$ and rephrasing the resulting expression. Then, the transformation between $n+(m+A)$ and $(n + m) + A$ is a simple rearrangement of brackets. We omit the details. □

By combining the transitivity of many-one equivalence and the commutativity of the addition of natural numbers with Theorem 3.1.6, we obtain a commutativity-like property of "+"in the following sense.

Corollary 3.1.7. *Let A be a language from* NP*. Then for each pair of natural numbers n and m, $n + (m + A)$ and $m + (n + A)$ are many-one equivalent via linear-time functions.*

3.2 First hardness results by auto-reductions

In this section we give some first results, which are based on a simple idea. When we manage to reduce an NP-complete problem A to the alternative solution problem $n{+}A$, which is by Theorem 3.1.5 also in NP, then we have the NP-completeness of $n + A$. We call this kind of reduction an *auto-reduction*. The idea behind all reductions is to add n new solutions to a given instance without affecting its original solutions. As an example, we apply this idea to show the NP-completeness of the ASP's for Kernel, Minimum Edge Cost Flow, and Subset Sum. To reduce redundancy in commentarial sentences, we only give the definition of the treated problems and the used verifiers without further explanations.

3.2.1 Kernel

Problem Description (Kernel)

Given: A digraph $G = (V, A)$.

Question: Does G have a kernel, i.e., is there a set $S \subseteq V$ such that there is no arc between any two vertices from S and for each vertex $u \in V \setminus S$ there is a kernel vertex $v \in S$ such that (u, v) is an arc in A ?

Standard Verifier:

$$V_{\text{Kernel}} = \{(G, S) : G \text{ is a digraph and } S \text{ is a kernel of } G\}.$$

Theorem 3.2.1. *For each $n \in \mathbb{N}^+$, $n +$ Kernel is* NP*-complete.*

Proof. We will give a many-one reduction Kernel $\leq_m^p$ $n +$ Kernel, for each natural number n. Since Kernel was shown to be NP-complete in [Chv73], the claim follows immediately. Let $n \in \mathbb{N}^+$ and let $G = (V, A)$ be an instance for Kernel.

We define a graph $G' = (V', A')$ based on $G = (V, A)$ as follows: Add n new vertices $k_1, k_2, \ldots, k_n$ to V, i.e., $V' = V \cup \{k_1, k_2, \ldots, k_n\}$. For each new vertex k_i, add the arcs $\{(k_i, v) : v \in V\}$ and $\{(v, k_i) : v \in V\}$ to A. Finally, we add all of the $n(n-1)$ possible arcs between the n new vertices. So

$$\begin{aligned} A' =& A \cup \{(k_i, v) : v \in V \wedge 1 \leq i \leq n\} \cup \{(v, k_i) : v \in V \wedge 1 \leq i \leq n\} \\ & \cup \ \{(k_i, k_j) : 1 \leq i, j \leq n \wedge i \neq j\}. \end{aligned}$$

Observe that all the sets $\{k_i\}, 1 \leq i \leq n$, are kernels of $G' = (V', A')$. A kernel consisting of two or more elements can not contain one of the n new vertices, since each new vertex is joined by an arc with all other vertices in V'. Hence, every kernel of G' different from the n kernels above is contained in the original vertex set V. It is easy to see that such a kernel is also a kernel of G and, conversely, a kernel of G is also a kernel of G'. Hence, the polynomial-time computable mapping $G \mapsto (G', k_1, k_2, \ldots, k_n)$ realizes the desired reduction from Kernel to $n +$ Kernel. □

3.2.2 Minimum Edge Cost Flow (MECF)

Problem Description (Minimum Edge Cost Flow)

Given: A 7-tuple (G, s, t, c, p, R, B), where $G = (V, A)$ is a digraph, s (source) and t (sink) are specified vertices, c (capacity) and p (price) are mappings from A to $\mathbb{N}^+$, and R, B are positive integers.

Question: Is there a flow function $f : A \to \mathbb{N}$ such that

1. $f(a) \leq c(a)$ for all $a \in A$,
2. for each $v \in V \setminus \{s, t\}$, $\sum_{(u,v)\in A} f((u, v)) = \sum_{(v,u)\in A} f((v, u))$, i.e., flow is conserved in v,
3. $\sum_{(u,t)\in A} f((u, t)) - \sum_{(t,u)\in A} f((t, u)) \geq R$, i.e., the net flow into t is at least R, and
4. if $A' = \{a \in A : f(a) \neq 0\}$, then $\sum_{a\in A'} p(a) \leq B$?

Standard Verifier:

$V_{\text{MECF}} = \{((G, s, t, c, p, R, B), f) : G = (V, A)$ is a digraph, $s, t \in V$, c and p are mappings from A to $\mathbb{N}^+$, $R, B \in \mathbb{N}^+$, and f is a flow function with the above properties$\}$.

Theorem 3.2.2. *For each $n \in \mathbb{N}^+$, $n +$* Minimum Edge Cost Flow *is* NP*-complete.*

Proof. Let $n \in \mathbb{N}^+$. We prove the NP-hardness of $n+$MECF by a reduction from MECF, whose NP-completeness was shown in [EJ77]. Let (G, s, t, c, p, R, B) be an instance for MECF. We add n new vertices $x_1, x_2, \ldots, x_n$ and the $2n$ new arcs $\{(s, x_i) : 1 \leq i \leq n\} \cup \{(x_i, t) : 1 \leq i \leq n\}$ to G and let G' denote the new graph. The capacity of all new arcs is R. The price is $\lfloor B/2 \rfloor$ for the arcs $\{(s, x_i) : 1 \leq i \leq n\}$ and $\lceil B/2 \rceil$ for the arcs $\{(x_i, t) : 1 \leq i \leq n\}$. Let c' and p' denote the changed capacity and price functions of G'.

The n flow functions $f_1, f_2, \ldots, f_n$ that map the arcs $(s, x_i), (x_i, t)$ to R and all other arcs to zero are solutions for the modified instance (G', s, t, c', p', R, B). Using one pair of new arcs $((s, x_i), (x_i, t))$ costs B, hence the n above flow functions are the only possible ones using the new arcs. So it is obvious that there is a solution for (G', s, t, c', p', R, B) different from $f_1, f_2, \ldots, f_n$ if and only if there exists a solution for (G, s, t, c, p, R, B). Hence, the polynomial-time computable mapping $(G, s, t, c, p, R, B) \mapsto ((G', s, t, c', p', R, B), f_1, f_2, \ldots, f_n)$ realizes the reduction. □

3.2.3 Subset Sum (SS)

Problem Description (Subset Sum)

Given: A finite set A, a size function s from A to $\mathbb{N}^+$, and a positive integer B.

Question: Is there a subset $A' \subseteq A$ such that $\sum_{a \in A'} s(a) = B$?

Standard Verifier:

$$V_{\text{SS}} = \{((A, s, B), A') : A \text{ is a finite set, } s \text{ is a mapping from } A \text{ to } \mathbb{N}^+, B \text{ is from } \mathbb{N}^+, \text{ and } A' \text{ is a subset of } A \text{ such that } \sum_{a \in A'} s(a) = B\}.$$

Theorem 3.2.3. *For each $n \in \mathbb{N}^+$, $n +$ Subset Sum is* NP*-complete.*

Proof. Let n be some natural number. We give a many-one reduction from Subset Sum to $n +$ Subset Sum. The needed fact that Subset Sum is NP-complete was shown in [Kar72]. Let (A, s, B) be an instance for Subset Sum.

We define a new instance for Subset Sum as follows: Add n new elements $b_1, \ldots, b_n$ to A. Denote this new set by A'. Define the new size-function $s'(a) = s(a)$, for all $a \in A$, and $s'(b_i) = B$, for all $1 \leq i \leq n$. Obviously, all subsets $\{b_i\}, 1 \leq i \leq n$, are solutions for $(A \cup \{b_1, \ldots, b_n\}, s', B)$. It is also not hard to see that there are no further solutions that contain one of the elements $b_1, \ldots, b_n$. Hence, each additional solution of $(A \cup \{b_1, \ldots, b_n\}, s', B)$ is also a solution of (A, s, B) and vice versa. Thereore, the function that maps (A, s, B) to $((A', s', B), \{b_1\}, \ldots, \{b_n\})$ realizes the desired reduction from Subset Sum to $n +$ Subset Sum. □

3.3 Generating parsimonious reduction

So far, we have shown the NP-completeness of the alternative solutions problems $n + A$, $n \in \mathbb{N}^+$, for several NP-complete problems A, by giving a reduction $A \leq_m^p n + A$, for all $n \in \mathbb{N}^+$. Surely, we could follow this idea and discuss a lot more NP-complete problems in the same way. Since this approach provides no deeper insight into the field of alternative solutions, it appears to be unsatisfactory. Instead we want to find ways to discuss the complexity of alternative solutions that provide a better understanding of this area.

As a first step, in this section we try to find a notion of a reduction between NP-problems with associated verifiers that also provides a connection between the complexity of alternative solutions of the treated problems. In particular, we would like to have a result of the form

$$(A, V_A) \leq_? (B, V_B) \rightarrow (\forall n \in \mathbb{N}^+)[n + A_{V_A} \leq_m^p n + B_{V_B}], \tag{3.1}$$

for a sought-after type of reduction $\leq_?$.

It turns out, that the generating parsimonious reduction, which is a many-one-reduction that furthermore provides a (generating) function that transforms solutions between the involved problems, is such a type of reduction. For the introduction of the generating parsimonious reduction, we refer to the next section. Note that the generating parsimonious reduction will also help to give easier auto-reductions.

3.3.1 Basic definitions and properties

Here, we give the definition of the generating parsimonious reduction and gather some basic properties.

Definition 3.3.1 ([LL78]). *Let A and B be problems from* NP *with associated verifiers V_A and V_B. We call (A, V_A)* generating parsimoniously *reducible (gp-reducible) to (B, V_B) ($(A, V_A) \leq^p_{gp} (B, V_B)$) if and only if there exists a pair (f, g) of polynomial-time computable functions $f : \Sigma^* \to \Sigma^*$ and $g : \Sigma^* \times \Sigma^* \to \Sigma^*$ such that for all $x \in \Sigma^*$:*

1. *$x \in A \leftrightarrow f(x) \in B$ and*
2. *g_x, defined via $g_x(y) = g(x, y)$, is a bijection between $V_A(x)$ and $V_B(f(x))$.*

Note that the first requirement follows from the second one. However, we explicitly state it to point out that *gp*-reducibility implies many-one reducibility. Since the function g_x generates solutions for $f(x)$ from solutions for x, we call g a *generating function*. Note furthermore that the reduction depends on the choice of the verifiers V_A and V_B.

It is immediate that the *gp*-reduction is reflexive, since $(A, V_A) \leq^p_{gp} (A, V_A)$ via $f = id$ and $g_x = id$, for all $x \in \Sigma^*$. The transitivity of the *gp*-reduction is stated in the following lemma.

Lemma 3.3.2 ([LL78]). *The gp-reduction is transitive.*

Proof. Let $A_i, i \in \{1, 2, 3\}$, be from NP with respective verifiers V_{A_i}.

Let $A_1 \leq^p_{gp} A_2$ via (f^1, g^1) and $A_2 \leq^p_{gp} A_3$ via (f^2, g^2) and let x be an arbitrary instance of A_1. So f^1 maps x to an instance $f^1(x)$ of A_2 such that g^1_x is a bijection between $V_{A_1}(x)$ and $V_{A_2}(f^1(x))$. Analogously, f^2 maps $f^1(x)$ to the instance $f^2(f^1(x))$ of A_3 such that $g^2_{f^1(x)}$ is a bijection between $V_{A_2}(f^1(x))$ and $V_{A_3}(f^2(f^1(x)))$. Since the composition of two bijective functions is also bijective, $g^2_{f^1(x)} \circ g^1_x$ is a bijection between $V_{A_1}(x)$ and $V_{A_3}(f^2(f^1(x)))$. Due to the fact that the composition of two polynomial-time computable functions is also polynomial-time computable, we have that $A_1 \leq^p_{gp} A_3$ via $(f^2 \circ f^1, g^2_{f^1(x)} \circ g^1_x)$. □

In Theorem 3.1.6 and Corollary 3.1.7 it was stated that $n + (m + A)$ and $(n + m) + A$ as well as $n + (m + A)$ and $m + (n + A)$ are many-one equivalent via realtime functions. Having a closer look at the proofs, it can easily be seen that both assertions can be slightly strengthened by using the *gp*-reduction. In both cases, all we have to do is to add the identity as generating function g_x, for all $x \in \Sigma^*$.

Corollary 3.3.3. *Let A be a language from* NP*. Then $n+(m+A)$ and $(n+m)+A$ are gp-equivalent*[1]*, for all natural numbers n and m via linear-time functions.*

Corollary 3.3.4. *Let A be a language from* NP*. Then for each pair of natural numbers n and m, $n + (m + A)$ and $m + (n + A)$ are gp-equivalent via linear-time functions.*

3.3.2 Easier auto-reductions using *gp*-reductions

In Section 3.2 we have shown the NP-completeness of all $n + A$ for several NP-complete problems A by giving infinitely many reductions $A \leq^p_m n + A$, $n \geq 1$. In this section we will see that proving one *gp*-reduction $A \leq^p_{gp} 1 + A$ implies the same result. Since a reduction $A \leq^p_{gp} 1 + A$ produces an instance with one additional solution with respect to V_A it is not surprising that repeated application of this reduction produces an instance with several additional solutions, which is the basic idea of the upcoming proof.

Theorem 3.3.5. *Let A be a problem in* NP *and let V_A be a verifier for A. If A is gp-reducible to $1 + A$, then A is gp-reducible to $n + A$, for each natural number n.*

Proof. Let V_{n+A} denote the standard verifier for $n + A$ based on V_A as defined in the proof of Theorem 3.1.5. Let furthermore, $A \leq^p_{gp} 1 + A$ via (f, g).

We prove the theorem by induction over n. For $n = 0$ and $n = 1$ the claim is trivial. Let $A \leq^p_{gp} n + A$ for a fixed natural number n. To conclude that $A \leq^p_{gp} (n + 1) + A$, it suffices to show that $n + A$ is generating parsimoniously reducible to $(n + 1) + A$ via some functions f' and g', because of the transitivity of *gp*-reductions.

Let $z = (x, y_1, \ldots, y_n)$ be an instance for $n + A$. Let $f(x) = (x', y')$. Then, we define $f'(z) := (x', y', g_x(y_1), \ldots, g_x(y_n))$ and $g'_z(y) := g_x(y)$, for all $y \in \Sigma^*$. Since f and g are polynomial-time computable, both functions f' and g' are also in FP. Moreover, the property that g is injective on $V_A(x)$ carries over to g' on $V_{n+A}(z) \subseteq V_A(x)$. It remains to show that $g'_z(V_{n+A}(z)) = V_{(n+1)+A}(f'(z))$, that is, g'_z maps the set of solutions (w.r.t. V_A) for x different from $y_1, \ldots, y_n$ onto the set of solutions (w.r.t. V_A) for x' different from $y', g_x(y_1), \ldots, g_x(y_n)$.

[1] Analogous to many-one equivalence, *gp*-equivalence stands for *gp*-reducibility in both directions.

In order to show that $g'_z(V_{n+A}(z)) \subseteq V_{(1+n)+A}(f'(z))$ let $y \in V_{n+A}(z)$. It follows that $y_1, \ldots, y_n$, and y are pairwise different solutions for x w.r.t. V_A. Hence, $g_x(y_1), \ldots, g_x(y_n)$, and $g_x(y)$ are pairwise different solutions for $f(x) = (x', y')$ w.r.t. V_{1+A} because $A \leq^p_{gp} 1 + A$ via (f, g). By the definition of $(n + 1) + A$, we have that $g_x(y) = g'_z(y) \in V_{1+n+A}(f'(z))$.

For the converse direction assume that $s \in V_{1+n+A}(f'(z))$. It follows that $g_x(y_1), \ldots, g_x(y_n)$, and s are pairwise different solutions for (x', y') w.r.t. V_{1+A}. Since g_x is a bijection between $V_A(x)$ and $V_{1+A}((x', y'))$, there exists a string $y \in V_A(x)$ such that $g_x(y) = s$ and y differs from $y_1, \ldots, y_n$. Hence, $y \in V_{n+A}(z)$ and thus, $s \in g_x(V_{n+A}(z)) = g'_z(V_{n+A}(z))$. □

3.3.3 Hardness results by auto-*gp*-reductions

In the previous section we stated that giving only one *gp*-reduction $1 + A \leq^p_{gp} A$ for an NP-complete problem A is sufficient to prove the NP-completeness for all $n+A$, $n \in \mathbb{N}^+$ (Theorem 3.3.5). As an example we will show the NP-completeness of n+SAT and n+PARTITION by giving one auto-*gp*-reduction.

Satisfiability

Problem Description (Satisfiability)

Given: A formula F in CNF.

Question: Is F satisfiable ?

Standard Verifier:

$$V_{\text{SAT}} = \{(F, \alpha) : F \text{ is a CNF-formula and } F(\alpha) = 1\}.$$

The following result was firstly stated in [YS02].

Theorem 3.3.6. *For each $n \in \mathbb{N}^+$, $n + \text{SAT}$ is* NP*-complete.*

Proof. We prove the claim by giving a generating parsimonious reduction from SAT to $1 + \text{SAT}$. Let F be an instance for SAT, i.e., a Boolean formula in CNF on the variable set $V = \{x_1, \ldots, x_n\}$ and let $C_1, C_2, \ldots, C_m$ denote the clauses of F.

Based on F, we describe a formula F' over the variable set $V' = V \cup \{y\}$. The formula F' consists of the modified clauses $C'_i = C_i \vee y$ and of the additional clauses $C'_{m+i} = (\neg y \vee x_i)$, for $1 \leq i \leq n$. Observe that an assignment β with $\beta(y) = 1$ satisfies F' if and only if $\beta(y) = \beta(x_1) = \cdots = \beta(x_n) = 1$.

Observe furthermore, that an assignment β' with $\beta'(y) = 0$ satisfies F' if and only if β' satisfies the clauses $C'_1, C'_2, \ldots, C'_m$, that is, the restriction of β' to the variable set V satisfies F.

Now, let f denote the assignment that maps F to (F', one), where *one* denotes the assignment $one(z) = 1$, for all $z \in V'$. Let furthermore g_F denote the function that maps an assignment $\alpha : V \to \{0,1\}$ to the assignment $\alpha' : V \cup \{y\} \to \{0,1\}$ as follows:

$$\alpha'(z) = \begin{cases} 0, & \text{if } z = y \\ \alpha(z), & \text{otherwise} \end{cases}.$$

By the above observations we have that the set of satisfying assignments of F' is $V_{\mathrm{SAT}}(F') = \{one\} \cup g_F(V_{\mathrm{SAT}}(F))$, that is, $\mathrm{SAT} \leq^p_{gp} 1 + n$ via f and g. □

Partition

Problem Description (Partition)

Given: A finite multiset A of positive integers.

Question: Is there a partition $\{A', A''\}$ of A such that $\sum_{k \in A'} k = \sum_{k \in A''} k$?

Standard Verifier:

$$V_{\textsc{Partition}} = \{(A, \{A', A''\} : A \text{ is a finite multiset of positive integers and } \{A', A''\} \text{ is a partition of } A \text{ such that } \sum_{k \in A'} k = \sum_{k \in A''} k\}.$$

Note that the only difference between sets and multisets is the fact that an element of a multiset additionally has a multiplicity, which is a natural number, indicating how many memberships the element has in the multiset. The join $A \uplus B$ of two multisets A, B is composed by adding the multiplicities of the members from A and B. For instance, the join of $\{1,2,3\}$ and $\{2,3,4\}$ is $\{1,2,2,3,3,4\}$. While computing a sum over a multiset A each element appears k times, if k is its multiplicity.

The given definitions of Partition and $V_{\textsc{Partition}}$ might appear more complicated than the more common ones (e.g. see [GJ79]), where Partition typically consists of a set A of elements and a size-function s, assigning a natural number to each element. A solution in this sense, is a set $A' \subset A$ such that $\sum_{a \in A'} s(a) = \sum_{a \in A \setminus A'} s(a)$. From our point of view, this definition has a drawback. Suppose that there are two elements $a_1, a_2 \in A$ having the same weight and let A' be a solution such that $a_1 \in A'$ and $a_2 \notin A'$. Against our intuitive understanding of solutions, there are alternative solutions $A' \setminus \{a_1\} \cup \{a_2\}$ and $A \setminus A'$ that actually describe the same situation. Since our definition avoids such trivial alternative solutions, it seems to be more practical for studying alternative solutions.

Theorem 3.3.7. *For each $n \in \mathbb{N}^+$, n + Partition defined with respect to $V_{\textsc{Partition}}$ is* NP*-complete.*

Proof. We give a *gp*-reduction PARTITION $\leq^p_{gp}$ $1 +$ PARTITION. Since PARTITION was shown to be NP-complete in [Kar72], the theorem follows immediately from Theorem 3.3.5. Let A be an instance for PARTITION and let S denote the sum $\sum_{a \in A} a$. Note that if S is odd then $A \notin$ PARTITION. So suppose S is even. If A includes an element $a = S/2$ there is exactly one solution, namely the partition $\{\{a\}, A \setminus \{a\}\}$. Such instances are mapped to a fixed member of $1 +$ PARTITION having one solution. In this case, the partition $\{\{a\}, A \setminus \{a\}\}$ is mapped to this solution.

In the following assume that $a \neq S/2$, for all $a \in A$. We define $f(A) := (A \biguplus \{S/2, S/2\}, \{A, \{S/2, S/2\}\})$. It is easy to see that $\{A, \{S/2, S/2\}\}$ is a partition for $(A \biguplus \{S/2, S/2\})$. For each alternative partition $\{A_1, A_2\}$, it holds that $S/2 \in A_1$ and $S/2 \in A_2$. Hence, the function g that maps a partition $\{A', A''\}$ of A to the partition $\{A' \biguplus \{S/2\},\ A'' \biguplus \{S/2\}\}$ of $A \biguplus \{S/2, S/2\}$ is a bijection between the partitions of A and the partitions of $A \biguplus \{S/2, S/2\}$ that differ from $\{A, \{S/2, S/2\}\}$. Thus, we have PARTITION $\leq^p_{gp}$ $1 +$ PARTITION via f and g. □

3.3.4 Connected subgraph problems

This short section deals with a class of problems for which the completeness of the ASP's is very easy to see, so called *connected subgraph problems*. The antetype for this class is the problem CLIQUE:

Problem Description (Clique)

Given: A graph $G = (V, E)$ and a natural number k.

Question: Is there a clique of size at least k in G, i.e., is there a set C of vertices such that $G[C]$ is isomorphic to the complete graph $K_{|C|}$?

Standard Verifier:

$$V_{\text{CLIQUE}} = \{((G, k), C) : G \text{ is a graph, } k \in \mathbb{N}, \text{ and } C \text{ is a } k\text{-clique of } G\}.$$

It is very easy to see that CLIQUE $\leq^p_{gp}$ $1 +$ CLIQUE, since the mapping

$$f : (G, k) \mapsto ((G \cup K_k, k), K_k)$$

combined with $g = id$ realizes the reduction. By Theorem 3.3.5 it follows that $n + A$ is NP-complete, for each natural number n.

It is obvious, that the same idea also works for problems like LONGEST CYCLE, LONGEST PATH, or INDUCED PATH that ask for connected subgraphs with a local property that is independent of the rest of the graph.

Theorem 3.3.8. *For each* $n \in \mathbb{N}^+$, $n +$ CLIQUE, $n +$ LONGEST CYCLE, $n +$ LONGEST PATH, *and* $n +$ INDUCED PATH *are* NP*-complete.*

3.3.5 Inter-*gp*-reduction

The starting point for the discussion of *gp*-reductions was the wish for a reduction $\leq_?$ between two NP-problems A and B with associated verifiers V_A and V_B such that $(A, V_A) \leq_? (B, V_B)$ implies $n + A \leq_m^p n + B$, for all positive natural numbers n.

The following theorem states that the *gp*-reduction actually is such a reduction. It is even slightly stronger, because we get *gp*-reducibility between $n + A$ and $n + B$. Since we can transfer the hardness of the ASP's from one NP-complete problem to another, we call this type of reduction an *inter-reduction*.

Theorem 3.3.9. *Let A and B be problems in* NP *with associated verifiers V_A and V_B. If $A \leq_{gp}^p B$ then it holds that $n + A \leq_{gp}^p n + B$, for each natural number n.*

Proof. First, recall the standard verifier for $n + A$, that was defined in the proof of Theorem 3.1.5. If V_A is a natural verifier for A then we use the verifier

$$V_{n+A} = \{((x, y_1, \ldots, y_n), y) : (\forall\ 1 \leq i \leq n)[(x, y_i) \in V_A], (\forall\ 1 \leq i \neq j \leq n)[y_i \neq y_j], \\ (\forall\ 1 \leq i \leq n)[y_i \neq y] \wedge (x, y) \in V_A\}.$$

as standard verifier for $n + A$. Analogous, we get the standard verifier V_{n+B} for $n + B$.

Note that the assertion is trivial for $n = 0$. Now, let $A \leq_{gp}^p B$ via (f, g) and let $n \in \mathbb{N}^+$. Recall that $A \leq_{gp}^p B$ via (f, g) means that f maps an instance x for A to $f(x)$ such that g_x is a bijection between $V_A(x)$ and $V_B(f(x))$.

To show that $n + A \leq_{gp}^p n + B$, we will give functions f' and g' based on f and g that realize this reduction. Let f' be the function that maps a given instance $z = (x, y_1, y_2, \ldots, y_n)$ for $n + A$ to $f'(z) = (f(x), g_x(y_1), g_x(y_2), \ldots, g_x(y_n))$. It is easy to see that if $y_1, y_2, \ldots, y_n$ are no pairwise different solutions for x then $g_x(y_1), g_x(y_2), \ldots, g_x(y_n)$ are no pairwise different solutions for $f(x)$ and thus in this case, $V_{n+A}(z) = V_{n+B}(f(z)) = \varnothing$.

So suppose that all y_i are pairwise different solutions for x. Since g_x is a bijection between $V_A(x)$ and $V_B(f(x))$ as well as between $\{y_1, y_2, \ldots, y_n\} \subseteq V_A(x)$ and $\{g_x(y_1), g_x(y_2), \ldots, g_x(y_n)\} \subseteq V_B(f(x))$, g_x is a bijection between $V_{n+A}(z) = V_A(x) \setminus \{y_1, y_2, \ldots, y_n\}$ and $V_{n+B}(f'(z)) = V_B(f(x)) \setminus \{g_x(y_1), g_x(y_2), \ldots, g_x(y_n)\}$.

So by defining $g'_z(y) := g_x(y)$, we gain $n + A \leq_{gp}^p n + B$ via f' and g'. □

Note that we now have a very simple proof for Theorem 3.3.5 by combining Theorem 3.3.9 with Corollary 3.3.3.

For hardness results by inter-reduction we refer to Chapter 4, where the question of the complexity of alternative solutions is embedded in a discussion about the "right" verifier for NP-problems.

A closer look at the proof of the NP-completeness of SAT [Coo71] shows that cook implicitly proved the following stronger theorem that states that SAT is also complete for NP with respect to *gp*-reductions.

Theorem 3.3.10. *[LL78] If A is a language from* NP *with a verifier V_A then (A, V_A) is generating parsimoniously reducible to $(\mathrm{SAT}, V_{\mathrm{SAT}})$, where*

$$V_{\mathrm{SAT}} = \{(F, \alpha) : F \text{ is a CNF-formula and } F(\alpha) = 1\}.$$

Proof. Let A be a language from NP with a verifier V_A and let p_A be the polynomial that bounds the length of solutions y for each instance x, i.e., for each pair (x, y) such that $(x, y) \in V_A$, we have $|y| \leq p_A(|x|)$. We consider the nondeterministic machine M' that given an input string x, in a first step guesses all strings y with $|y| \leq p_A(|x|)$ and then deterministically verifies if $(x, y) \in V_A$. So each accepting path of the computation of $M'(x)$ represents a solutions y for the instance x and for each solution y of x there is an accepting path of $M'(x)$. This yields a bijection between the accepting paths of $M'(x)$ and the solutions of x. Applying Cook's construction to the machine M' and the input x provides a formula $f_{M'}(x) = F$ such that each satisfying assignment of F corresponds to exactly one accepting path of the computation of $M'(x)$ and vice versa. As seen above, there is also a bijection between the accepting paths of the computation of M' on the input x and the solutions $V_A(x)$. Combining both bijections, we gain a bijection g_x between $V_A(x)$ and the satisfying assignments $V_{\mathrm{SAT}}(F)$ of $f_{M'}(x) = F$. Hence, $(A, V_A) \leq^p_{gp} (\mathrm{SAT}, V_{\mathrm{SAT}})$ via $f_{M'}$ and g. □

Later (in Chapter 4), we will benefit from this theorem in the context of universal verifiers.

3.4 Bibliographical remarks

As already mentioned in the beginning of this chapter, some of the mentioned concepts have been introduced similarly by Ueda and Nagao [UN96, Set02, YS02]. They were especially interested in alternative solutions of puzzles like, e.g., Fillomino, Kakkuro, Nonogram and Sudoku, because puzzles shall have unique solutions and thus, the question for alternative solutions is crucial for puzzles.

In order to discuss these problems, the group of Ueda and Nagao very similarly defined the notions of ASP's for NP-problems 3.1.4, showed one direction of Corollary 3.1.6 (associativity of "+") and introduced the *gp*-reduction (they call it ASP-reduction) in order to show equivalents of Theorem 3.3.5 (auto-reduction) and Theorem 3.3.9 (inter-reduction). Using these notions and properties, they showed the NP-completeness of the ASP's of Hamiltonian Cycle, Satisfiability, 3Satisfiability, and 3Dimensional Matching which were used to show that the ASP's of the puzzles Fillomino, Kakkuro, Nonogram, Slither Rink, and Sudoku are NP-complete.

Unfortunately, in the beginning of our work on this thesis we did not know about the mentioned Japanese group and their results and redeveloped some of these concepts independently. One could say that this degrades the corresponding parts of our thesis. We rather consider this as a substantiation of our results. We

designed a very new formalization and found that another group with a very similar objective came to almost the same notions and concepts independently, which supports our ideas.

To the best of our knowledge the results not mentioned above, are new.

3.5 Conclusions

We have provided several ways to show NP-completes of all ASP's for an NP-complete problem. There are two types of auto-reductions, that is, on the one hand giving infinitely many auto-many-one reductions (one per ASP) and on the other hand giving one auto-*gp*-reduction. Furthermore, one can identify a given problem as a connected (induced) subgraph problem or show the NP-completeness for all ASP's by giving one inter-*gp*-reduction. As examples for these concepts, we have shown that all alternative solution problems of Kernel, Minimum Edge Cost Flow, Subset Sum, Partition, Clique, Induced Path, Longest Cycle, and Longest Path are NP-complete. The group of Ueda and Nagao has furthermore shown, that the ASP's of Hamiltonian Cycle, Satisfiability, 3Satisfiability, 3Dimensional Matching, and of the puzzles Fillomino, Kakkuro, Nonogram, Slither Rink, and Sudoku are also NP-complete. This list[2] gives reason to conjecture that all alternative solution problems of NP-complete problems are NP-complete, when defined with respect to their natural verifier. However, since the notion of a natural verifier is not formally definable, we abstain from formulating and discussing this conjecture and refer to the next chapter, where we replace the informal notion of a natural verifier by the notion of a universal verifier.

Moreover, we have seen that Cook's proof for the NP-completeness of SAT can be transferred to *gp*-reductions, that is, each NP-problem with any verifier is *gp*-reducible to $(\mathrm{SAT}, V_{\mathrm{SAT}})$. This will be of major importance in the next chapter.

Note that all hardness results of this chapter will recur in the next chapter as a by-product of showing that the used natural verifiers are universal verifiers.

[2]Anticipant, we say that we treat even more problems in the next chapter and show NP-completeness for their alternative solution problems.

Chapter 4

Alternative Solutions in NP II - Universal Verifiers and $\exists_r \forall_l gp$-Reduction

In the previous chapters we always used the natural verifier to characterize the solutions for an instance of an NP-problem. Although the choices of the used verifiers seemed to be canonical, using exactly these verifiers has no formal justification: It is possible, that there are different characterizations of given problems, leading to different natural verifiers. Or it might be the case that an NP-problem A has no canonical characterization of the form $A = \{x : (\exists y)|y| \leq p(|x|) \wedge B(x, y)\}$ with a P-problem B, that motivates the choice of B as the natural verifier. We are aware of the fact that for a very large majority of the known NP-problems, these doubts are baseless and there exist very canonical natural verifiers. However, we would prefer a formal characterization of proper verifiers, that on the one hand resolves all doubts about the choice of verifiers and on the other hand furthermore provides a possibility to formally state assertions of the form "For all NP-problems with the associated proper verifier it holds that...", which is in most cases impossible using the informal notion of natural verifiers.

A concept called *universal relations* that might be useful was introduced in [AB92]. The authors define properties called *joinability*, *couplability* and the existence of a so called *building block* for relations (verifiers) of NP-problems. Then, it is shown that all NP-problems B with any relation V_B can be reduced to each pair (A, V_A) of an NP-problem A with a relation V_A that satisfies these properties. Because of this universal reducibility, such relations are called universal relations. Unfortunately, in contrast to the concept of *gp*-reductions, the reduction used in [AB92] does not provide a bijection between solutions, but only a projection from one solutions set to another. It is not hard to see, that thus universal relations are not very useful for our strongly solution based approach. In this chapter we introduce the similar notion of universal verifiers based on the *gp*-reduction, which is a more promising candidate to replace the informal notion of the natural verifier.

We furthermore introduce a notion of reduction, the so-called $\exists_r \forall_l gp$-reduction, that is helpful in showing verifiers to be universal. We also prove that giving

such a reduction besides identifying the used verifier as universal also provides the NP-completeness of all ASP's defined with respect to this verifier. Then, we give such a reduction for a long list of NP-complete problems, showing their natural verifiers to be universal and answering our initial question by showing the NP-completeness of the associated alternative solution problems. So we will see that the formal notion of the universal verifier seems to be a proper substitute for the informal natural verifier, since almost all natural verifiers will turn out to be universal.

4.1 Definitions and properties

Many verifiers of NP-problems have the property that a given solution leads to one or more trivial alternative solutions. For instance, consider the verifier for Hamiltonian Cycle that identifies a Hamiltonian permutation of the vertices of a given graph G as a solution for G. Hence, each cyclic permutation of a given Hamiltonian permutation leads to another Hamiltonian permutation, which is another solution. Furthermore, it is not hard to modify a natural verifier such that the modified verifier induces trivial alternative solutions. For instance, let V_A be a natural verifier for A without trivial alternative solutions and let V'_A be the verifier that characterizes a string yz as a solutions for a given instance x if and only if $V_A(x, y)$ and $z \in \{0,1\}^k$. Hence, each solution with respect to V'_A is one of 2^k versions yz, $z \in \{0,1\}^k$, of one solution y with respect to V_A and there obviously are trivial alternative solutions. In the following we will use *w.r.t.* as abbreviation for *with respect to*.

Our idea of a proper verifier for an NP-problem is that such a verifier should reflect the intrinsic solutions of the problem (which is obviously also an informal phrase). So verifiers, where one known solutions leads to trivial alternative solutions reflecting the same intrinsic solution of the treated instances, shall be ruled out. In order to do so we concentrate on the numbers of solutions a certain verifier induces. Consider a verifier V that induces pairs of solutions such that one solution trivially leads to the other one. It follows that all instances have an even number of solutions. The same argumentation leads to the fact that the above constructed verifier V'_A induces a number of solutions divisible by 2^k. So we have seen, that such improper verifiers are restricted concerning their possible numbers of solutions. In contrast a proper verifier probably has a very general solution structure in terms of the possible numbers of solutions. Since the *gp*-reduction transfers some solution structure (at least the number of solutions), we feel that this reduction might be the right choice to distinguish between proper and improper verifiers. Let for instance B be an NP-problem with a verifier V_B with a general structure and a verifiers V'_B with an even number of solutions. Hence (B, V_B) is not *gp*-reducible to (B, V'_B) because an instance x with one solution w.r.t. V_B can not be mapped to an instance x' with one solution w.r.t. V'_B.

Following the presented intuition, we say that a verifier V_A for $A \in$ NP is

universal, if (A, V'_A) can be *gp*-reduced to (A, V_A), for all verifiers V'_A for A.

Definition 4.1.1. *Let A be from* NP*. A verifier V_A for A is called a* universal *verifier for A if and only if $(A, V'_A) \leq^p_{gp} (A, V_A)$ for each A-verifier V'_A.*

Note that the universal verifier for a language A is not unique. In fact, there might be many universal verifiers for the same language. For instance, if V_A is a universal verifier for A it is easy to show that

$$V'_A = \{(x, y1) : (x, y) \in V_A\}$$

is also universal. However, for each pair V_A and V'_A of universal verifiers for A holds $(A, V_A) \leq^p_{gp} (A, V'_A)$ and $(A, V'_A) \leq^p_{gp} (A, V_A)$, which implies that the associated ASP's have the same complexity (Theorem 3.3.9). So from our point of view different universal verifiers are basically the same[1].

Proving a verifier to be universal seems to be a very hard task, since there is an infinite number of verifiers for each NP-problem. The following notion of a reduction will finally help to show verifiers to be universal. This reduction is a slightly modified version of a verifier-independent witness isomorphic reduction from [FHT97].

Definition 4.1.2. *Let A and B be* NP*-problems. We call A $\exists_r\forall_l gp$-*reducible *to B ($A \leq_{\exists_r\forall_l gp} B$) if and only if there exists a verifier V_B for B such that for all verifiers V_A for A, it holds that $(A, V_A) \leq^p_{gp} (B, V_B)$. In this situation we say that A is $\exists_r\forall_l gp$-reducible to B via V_B.*

It is easy to see that A is $\exists_r\forall_l gp$-reducible to A via V_A if and only if V_A is universal for A.

When dealing with reductions, we often firstly discuss its properties as relation. As just mentioned the reflexivity of the $\exists_r\forall_l gp$-reduction is strongly connected with the existence of universal verifiers.

Corollary 4.1.3. *The $\exists_r\forall_l gp$-reduction is reflexive if and only if there exists a universal verifier for each problem from* NP*.*

Although we will later give universal verifiers for a lot of NP-complete problems, we do not know whether all NP-complete problems have universal verifiers or not. So the question, if the $\exists_r\forall_l gp$-reduction is reflexive, remains open. At least, we can show that the $\exists_r\forall_l gp$-reduction is transitive.

Lemma 4.1.4. *If $A \leq_{\exists_r\forall_l gp} B$ and $B \leq_{\exists_r\forall_l gp} C$ via V_C then $A \leq_{\exists_r\forall_l gp} C$ via V_C, i.e., the $\exists_r\forall_l gp$-reduction is transitive.*

Proof. Let $A \leq_{\exists_r\forall_l gp} B$ via V_B and $B \leq_{\exists_r\forall_l gp} C$ via V_C. Hence $(A, V_A) \leq^p_{gp} (B, V_B)$ for each A-verifier V_A and $(B, V_B) \leq^p_{gp} (C, V_C)$. Since generating parsimonious reductions are transitive (see Lemma 3.3.2) it holds that $(A, V_A) \leq^p_{gp} (C, V_C)$ for each A-verifier V_A. □

[1] In contrast, when comparing different universal verifiers concerning the complexity of their inverse problems (see e.g. [Che03]), they might behave different (see Chapter 8).

The following corollary is a simple conclusion from the definitions of universal verifiers and $\exists_r\forall_l gp$-reductions.

Corollary 4.1.5. *If $A \leq_{\exists_r\forall_l gp} B$ via V_B and $B \leq_{\exists_r\forall_l gp} A$ via V_A then V_A is universal for A and V_B is universal for B.*

Proof. By Lemma 4.1.4 we have $A \leq_{\exists_r\forall_l gp} A$ via V_A and $B \leq_{\exists_r\forall_l gp} B$ via V_B, which means that V_A is universal for A and V_B is universal for B. □

In analogy to many other reductions we define a notion of completeness for $\exists_r\forall_l gp$-reductions.

Definition 4.1.6. *A language A from a class $\mathcal{C}$ is called* $\exists_r\forall_l gp$-complete *for $\mathcal{C}$ if and only if $B \leq_{\exists_r\forall_l gp} A$, for all $B \in \mathcal{C}$.*

Since the class of our main interest is NP, we will use $\exists_r\forall_l gp$-complete as a shorthand for the term $\exists_r\forall_l gp$-complete for NP throughout this chapter.

Let us assume that A is a $\exists_r\forall_l gp$-complete problem and let B be an arbitrary NP-problem. It follows that B is $\exists_r\forall_l gp$-reducible to A via a verifier V_A^B for A that depends on B. Up to here, we can not expect that there exists a single verifier V_A for A such that all NP-problems are $\exists_r\forall_l gp$-reducible to A via that verifier V_A. If all problems from NP are $\exists_r\forall_l gp$-reducible to A via one and the same verifier V_A, we call *A $\exists_r\forall_l gp$-complete via V_A*.

The following theorem provides the existence of such a single verifier V_A for each $\exists_r\forall_l gp$-complete problem A.

Theorem 4.1.7. *Let $A \in$ NP be $\exists_r\forall_l gp$-complete. There exists a verifier V_A such that $B \leq_{\exists_r\forall_l gp} A$ via V_A, for all $B \in$ NP, that is, A is $\exists_r\forall_l gp$-complete via V_A.*

Proof. Let A be $\exists_r\forall_l gp$-complete. It follows that $A \leq_{\exists_r\forall_l gp} A$ via a verifier V_A. Let B be an arbitrary NP-problem. Since A is $\exists_r\forall_l gp$-complete, B is $\exists_r\forall_l gp$-reducible to A via a verifier V_A^B for A. By the fact that A is $\exists_r\forall_l gp$-reducible to A via V_A and by the transitivity of the $\exists_r\forall_l gp$-reduction (Lemma 4.1.4) it follows that $B \leq_{\exists_r\forall_l gp} A$ via V_A. □

It is a simple fact that the $\exists_r\forall_l gp$-completeness of a problem A via a verifier V_A implies that V_A is universal for A.

Observation 4.1.8. *If a problem A is $\exists_r\forall_l gp$-complete via V_A, then V_A is universal for A.*

Proof. Assume that A is $\exists_r\forall_l gp$-complete. It follows that A is $\exists_r\forall_l gp$-reducible to A via a verifier V_A. That verifier is by Definition 4.1.1 a universal verifier for A. □

Analogously to other reductions the upcoming lemma follows from the transitivity of $\exists_r\forall_l gp$-reducibility.

Lemma 4.1.9. *If $A \leq_{\exists_r\forall_l gp} B$ via V_B for a $\exists_r\forall_l gp$-complete problem A, then B is $\exists_r\forall_l gp$-complete (via V_B).*

Proof. Let A be $\exists_r\forall_l gp$-complete and let C be an arbitrary NP-problem. It follows that C is $\exists_r\forall_l gp$-reducible to A via an A-verifier V_A . Since A is $\exists_r\forall_l gp$-reducible to B via V_B, by the transitivity of the $\exists_r\forall_l gp$-reduction (Lemma 4.1.4), it follows that $C \leq_{\exists_r\forall_l gp} B$ via V_B. Thus, we have that B is $\exists_r\forall_l gp$-complete via V_B. □

Since we know that each $\exists_r\forall_l gp$-complete problem A has a universal verifier V_A (Observation 4.1.8) and that all problems from NP are $\exists_r\forall_l gp$-reducible to A via that verifier V_A (Theorem 4.1.7) we can slightly weaken the precondition of the above lemma.

Lemma 4.1.10. *Let A be a $\exists_r\forall_l gp$-complete problem with a universal verifier V_A and let B be a language from* NP *with a verifier V_B. If $(A, V_A) \leq^p_{gp} (B, V_B)$, it holds that B is $\exists_r\forall_l gp$-complete via V_B.*

Proof. Due to Lemma 4.1.9 it suffices to show that A is $\exists_r\forall_l gp$-reducible to B via V_B, i.e., $(A, V'_A) \leq^p_{gp} (B, V_B)$ for each verifier V'_A for A. So let V'_A be an arbitrary verifier for A. Since A is $\exists_r\forall_l gp$-complete via V_A we have that $(A, V'_A) \leq^p_{gp} (A, V_A)$. By the transitivity of the $\leq^p_{gp}$ reduction (Lemma 3.3.2) we have $(A, V'_A) \leq^p_{gp} (B, V_B)$. Thus, $A \leq_{\exists_r\forall_l gp} B$ via V_B. □

Hence, we can show $\exists_r\forall_l gp$-completeness for a problem B via an associated verifier V_B by giving one gp-reduction from (A, V_A) to (B, V_B), where A is $\exists_r\forall_l gp$-complete via V_A. So far, we have no such $\exists_r\forall_l gp$-complete problem. Thus, (a transcription of) the generalized version of Cook's Theorem 3.3.10 is of particular interest again, since it provides a first $\exists_r\forall_l gp$-complete problem.

Theorem 4.1.11 ([Coo71]). SAT *is $\exists_r\forall_l gp$-complete via V_{SAT}.*

Proof. In Theorem 3.3.10 we have seen that (A, V_A) is generating parsimoniously reducible to $(\mathrm{SAT}, V_{\mathrm{SAT}})$ for each NP-problem A and associated verifier V_A. In terms of $\exists_r\forall_l gp$-reduction that means that SAT is $\exists_r\forall_l gp$-complete via V_{SAT}. □

Note that Theorem 4.1.11 implies that V_{SAT} is a universal verifier for SAT and thus provides a first universal verifier.

Now, we can furthermore state a connection between the complexity of alternative solutions and the $\exists_r\forall_l gp$-reduction respectively $\exists_r\forall_l gp$-completeness that belatedly explains the importance of the class of $\exists_r\forall_l gp$-complete problems.

Theorem 4.1.12. *Let A be $\exists_r\forall_l gp$-complete via V_A. Then, $n + A$ (defined with respect to V_A) is* NP*-complete for all $n \in \mathbb{N}^+$.*

Proof. Let A be $\exists_r\forall_l gp$-complete via V_A. It follows that $(\mathrm{SAT}, V_{\mathrm{SAT}})$ is gp-reducible to (A, V_A) and hence, by Theorem 3.3.9, we have that $n+\mathrm{SAT} \leq^p_m n+A$ for all $n \in \mathbb{N}^+$, where $n + \mathrm{SAT}$ and $n + A$ are defined w.r.t. V_{SAT} and V_A. Thus, by the NP-completeness of $n+\mathrm{SAT}$, for all $n \in \mathbb{N}^+$ (see Theorem 3.3.6), the claim is immediate. □

By combining Observation 4.1.8 with Theorem 4.1.12, we immediately have the following corollary.

Corollary 4.1.13. *Let A be $\exists_r\forall_l gp$-complete via V_A. Then, V_A is universal for A and $n + A$ (defined with respect to V_A) is* NP*-complete for all $n \in \mathbb{N}^+$.*

So by showing $\exists_r\forall_l gp$-completeness for a problem A via a verifier V_A we answer two questions at once. We show that the used verifier V_A is universal for A and on the other hand, we show the NP-completeness of all alternative solution problems $n+A$, $n \in \mathbb{N}^+$, according to this universal verifier. In the remainder of this chapter we will show the $\exists_r\forall_l gp$-completeness of a lot of NP-complete problems via their natural verifiers. As we have seen in Observation 4.1.8, this implies that the used natural verifiers are universal verifiers. So we gain a formal justification for the use of these verifiers in characterizing the ASP's of the treated NP-problems. Moreover, Corollary 4.1.13 provides that all these ASP's are also NP-complete which negatively answers the initial question for the complexity of computing alternative solutions.

Before doing so, we have to keep a promise we gave in Chapter 3. When we formally introduced the ASP $n + B$ for an NP-problem B w.r.t. a verifier V_B, we also introduced the verifier

$$\begin{aligned} V_{n+B} =\{((x, y_1, \ldots, y_n), y) : (\forall\ 1 \leq i \leq n)[(x, y_i) \in V_B], (\forall\ 1 \leq i \neq j \leq n)[y_i \neq y_j], \\ (\forall\ 1 \leq i \leq n)[y_i \neq y] \wedge (x, y) \in V_B\}. \end{aligned}$$

as standard verifier for $n + B$ and claimed that for all treated problems B a universal verifier V_B for B leads to a universal verifier V_{n+B} for $n + B$. Since we will show the universality of the verifiers for the treated problems by giving gp-reductions from $\exists_r\forall_l gp$-complete problems (for instance SAT), it suffices to prove the following theorem.

Theorem 4.1.14. *If universality of V_B for B was shown by a gp-reduction from (A, V_A) to (B, V_B) for a $\exists_r\forall_l gp$-complete problem A with a universal verifier V_A, then V_{n+B} is also universal for $n + B$, for all $n \in \mathbb{N}^+$.*

Proof. Let $n \in \mathbb{N}^+$. By Theorem 4.1.11 we have that SAT is $\exists_r\forall_l gp$-complete via V_{SAT}. Furthermore, by the proof of Theorem 3.3.6 in combination with Theorem 3.3.5 we know that $(\mathrm{SAT}, V_{\mathrm{SAT}}) \leq^p_{gp} (n + \mathrm{SAT}, V_{n+\mathrm{SAT}})$, which implies that $n + \mathrm{SAT}$ is $\exists_r\forall_l gp$-complete via $V_{n+\mathrm{SAT}}$.

Due to the $\exists_r\forall_l gp$-completeness of A via V_A and the gp-reduction $(A, V_A) \leq^p_{gp} (B, V_B)$, we have that $(\mathrm{SAT}, V_{\mathrm{SAT}}) \leq^p_{gp} (B, V_B)$. By Theorem 3.3.9 it follows that $(n+\mathrm{SAT}, V_{n+\mathrm{SAT}}) \leq^p_{gp} (n+B, V_{n+B})$. Now, the $\exists_r\forall_l gp$-completeness of $n+B$ via V_{n+B} follows from the $\exists_r\forall_l gp$-completeness of $n+\mathrm{SAT}$ via $V_{n+\mathrm{SAT}}$ (Lemma 4.1.10). Thus, the assertion is immediate by Observation 4.1.8. □

4.2 The $\exists_r\forall_l gp$-completeness of the six basic NP-complete problems

In the previous section we provided a concept to give an exhaustive answer to the question of alternative solutions for NP-problems (Corollary 4.1.13) by giving one *gp*-reduction and thus showing $\exists_r\forall_l gp$-completeness. Namely, giving one *gp*-reduction $(A, V_A) \leq^p_{gp} (B, V_B)$ for a problem A that is $\exists_r\forall_l gp$-complete via V_A suffices to identify the verifier V_B as universal for B and to show that all $n + B$ defined with respect to the universal verifier V_B are NP-complete.

In this section, following this concept, we will discuss the six basic NP-complete problems listed in [GJ79], that are of especial interest as typical representatives for the class of NP-complete problems. In particular these problems are 3SAT, Hamiltonian Cycle, 3Dimensional Matching, Partition, Vertex Cover, and Clique. In order to show the $\exists_r\forall_l gp$-completeness of these problems we will establish and use $\exists_r\forall_l gp$-completeness results for some more problems.

Note that, since *gp*-reductions are special cases of many-one- or parsimonious reductions, in giving *gp*-reductions we can often benefit from known many-one- or parsimonious reductions that turn out to also be *gp*-reductions or that provide a basis for a *gp*-reduction.

To avoid unnecessary commentarial sentences, in most cases we only give the problem definition and the used verifier without further explanations.

4.2.1 3Satisfiability (3SAT)

Problem Description (3Satisfiability)

Given: A Boolean formula F in 3CNF.

Question: Is F satisfiable ?

Standard Verifier:

$$V_{3\text{SAT}} = \{(F, \alpha) : F \text{ is a Boolean 3CNF-formula and } F(\alpha) = 1\}.$$

The desired *gp*-reduction $(\text{SAT}, V_{\text{SAT}}) \leq^p_{gp} (3\text{SAT}, V_{3\text{SAT}})$ can be found in [YS02]. The following theorem is an immediate corollary of this reduction and Theorem 4.1.11.

Theorem 4.2.1. 3SAT *is* $\exists_r\forall_l gp$*-complete (via* $V_{3\text{SAT}}$*).*

4.2.2 Hamiltonian Cycle (HC)

Problem Description (Hamiltonian Cycle)

Given: A graph $G = (V, E)$.

Question: Is there a Hamiltonian cycle in G ?

Standard Verifier:

$$V_{\mathrm{HC}} = \{(G, C) : G \text{ is a graph and } C \text{ is a Hamiltonian cycle of } G\}.$$

In [Set02] the authors give a *gp*-reduction $(3\mathrm{SAT}, V_{3\mathrm{SAT}}) \leq^p_{gp} (\mathrm{HC}, V_{\mathrm{HC}})$ implying $\exists_r\forall_l gp$-completeness of HC. However, we give a very nice shorter proof that is inspired by the proof of Π^p_2-completeness of a Hamiltonian cycle problem in [KL95].

Theorem 4.2.2. Hamiltonian Cycle *is $\exists_r\forall_l gp$-complete via V_{HC}.*

Proof. Due to Lemma 4.1.10 it suffices to give a *gp*-reduction $(3\mathrm{SAT}, V_{3\mathrm{SAT}}) \leq^p_{gp} (\mathrm{HC}, V_{\mathrm{HC}})$. Let F be a 3CNF-formula consisting of the clauses $C_1 \wedge \cdots \wedge C_m$ over the variable set $X = \{x_1, \ldots, x_n\}$. The formula F will be mapped to a graph $f(F) = G_F$, which will be described in the following.

For each variable x_i, G_F contains a vertex x_i and for each clause C_j, there is a gadget H_j (see Figure 4.1) in G_F.

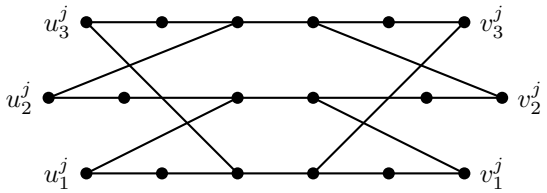

Figure 4.1: A depiction of the gadget H_j for the clause C_j.

The only vertices of the gadget H_j that are joining edges with the rest of the graph are the vertices u_1^j, u_2^j, u_3^j and v_1^j, v_2^j, v_3^j. By discussing all possible cases, it is not hard to see that each H_j has the following properties:

1. If a Hamiltonian cycle enters H_j in u_k^j, $k \in \{1, 2, 3\}$, then it leaves H_j in v_k^j.

2. For each $k \in \{1, 2, 3\}$, there is exactly one path from u_k^j to v_k^j that visits all vertices of H_j.

3. For all $1 \leq k_1 < k_2 \leq 3$, there is exactly one combination of two disjoint paths from $u_{k_1}^j$ to $v_{k_1}^j$ and from $u_{k_2}^j$ to $v_{k_2}^j$ such that all vertices of H_j are visited by one of the paths.

4. There is exactly one triple of disjoint paths P_1, P_2, and P_3 through H_j where P_ℓ leads from u_ℓ^j to v_ℓ^j, $1 \leq \ell \leq 3$, such that all vertices of H_j are visited by a path.

So, no matter how often and in which row(s) a potential Hamiltonian cycle tries to traverse the gadget, there is always a unique possibility to do so.

Now, we add edges such that there are two paths from x_1 to x_2. Finally, the choice between both paths will represent the assignment of the variable x_1.

To describe the first path, say x_1 appears in the k_1th, k_2th, ..., k_rth literal of the clauses $C_{j_1}, C_{j_2}, \ldots, C_{j_r}$, respectively. We add the edge $\{x_1, u_{k_1}^{j_1}\}$, the edges $\{v_{k_\ell}^{j_\ell}, u_{k_{\ell+1}}^{j_{\ell+1}}\}$, for all $1 \leq \ell \leq r-1$, and the edge $\{v_{k_r}^{j_r}, x_2\}$. So this path starts in x_1 traverses all gadgets H_j corresponding to those clauses that contain x_1 and finishes in x_2. This path shall represent assigning 1 to x_1. Analogously, we add a second path from x_1 to x_2 traversing those clauses which include $\neg x_1$. This path shall represent assigning 0 to x_1.

In the same way we add a pair of paths from x_2 to x_3, ..., from x_{n-1} to x_n and from x_n to x_1. This concludes the construction of the graph G_F.

So the basic idea of the construction is the following. A potential Hamiltonian cycle goes from x_i to x_{i+1}, for $1 \leq i \leq n-1$, and from x_n to x_1 always using one of the two possible paths. Note that the structure of the graph G_F and the properties of the gadget make sure that there is no Hamiltonian cycle of G_F that differs from this scheme. The used paths represent an assignment α of the variable set $X = \{x_1, \ldots, x_n\}$. No matter how often and where the cycle traverses some gadgets $H_1, \ldots, H_m$, there is always a unique way that visits all vertices of the gadgets. It is not hard to see that such a path visits the gadget H_j if and only if one of its literals is satisfied by α. Furthermore, it is obvious that such a cycle is a Hamiltonian cycle if and only if it traverses all gadgets, i.e., the associated assignment satisfies all clauses.

Hence, we have that G_F has exactly one Hamiltonian cycle C_α for each satisfying assignment α of F and conversely, each Hamiltonian cycle C of G_F induces an satisfying assignment α_C of F. So, when we define $g_F(\alpha) = C_\alpha$ we have $(\text{3SAT}, V_{\text{3SAT}}) \leq_{gp}^p (\text{HC}, V_{\text{HC}})$ via f and g, where $f(F) = G_F$. □

4.2.3 Tiling

In this section we will give a short introduction to tiling problems in order to finally show the $\exists_r\forall_l gp$-completeness of 3Dimensional Matching (For more detailed information about tiling problems we refer to [vEB97]). The geometrical character of a tiling problem T is often helpful in finding reductions from T to different kinds of problems. Since a certain type of a tiling problem will turn out to be $\exists_r\forall_l gp$-complete, we gain a useful possibility to prove $\exists_r\forall_l gp$-completeness of a problem A by giving a gp-reduction from this tiling problem to A. Finally we will use this method to show the $\exists_r\forall_l gp$-completeness of 3Dimensional Matching.

The most basic notion in tiling problems is a tile. A tile is a square which is divided into four triangles by the two diagonals. A certain coloring of the four triangles is called a tile type (see also Figure 4.2).

Definition 4.2.3 ([vEB97]). *Let C be a finite set of colors. A* tile type *d is a four-tuple (d_n, d_e, d_s, d_w) of colors from C. For a tile type $d = (d_n, d_e, d_s, d_w)$ we define $north(d) := d_n$, $east(d) := d_e$, $south(d) := d_s$, and $west(d) = d_w$.*

Following the idea of the colored triangles, we say that d_n, d_e, d_s, and d_w are the colors of the triangles in the north, east, south, and west, respectively.

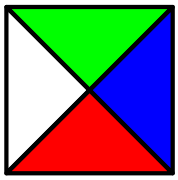

Figure 4.2: A depiction of a tile type with the colors $(green, blue, red, white)$.

Note that the tile types are orientated, that is, rotating or reflecting tile types in general leads to a different tile type

Tiling problems deal with the question, if a certain region R of the 2-dimensional grid can be tiled using a given set D of tile types (see Figure 4.3). That question leads to the notion of a D-tiling.

Definition 4.2.4 ([vEB97]). *Let R be a region of the two-dimensional square grid and let D be a set of tile types. A mapping $t : R \to D$ is called a* D-tiling *of R if and only if any two tiles that share a horizontal or vertical edge in the grid have the same color in the triangles adjacent to this edge.*

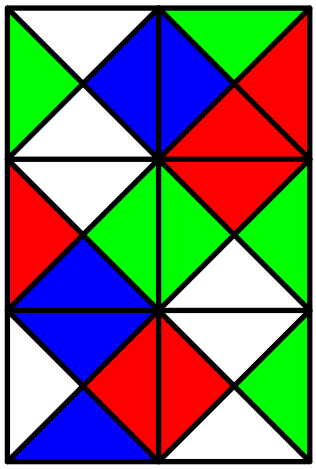

Figure 4.3: A proper tiling of the 2×3 grid using six different tile types.

So, a tiling problem is to decide, given a region R and a set of tile types D, whether a D-tiling of R exists, or not. Sometimes there are some additional

constrains, such as a given coloring of parts of the border of the region R, which has to be extended by a D-tiling. Depending on the choice of the range of regions R and the additional constraints tiling problems can be NP-complete, PSPACE-complete and even RE-complete [vEB97, Ber66].

We will focus on the following NP-complete tiling problem.

Problem Description (Tiling)

Given: A set of tile types D, a natural number n in unary representation and a coloring c of the upper and lower border of the $n \times n$ grid.

Question: Is there a D-tiling of the $n \times n$ grid, extending the given coloring on the border ?

Standard Verifier:

$V_{\text{TILING}} = \{((D, n, c), t) : D$ is a set of tile types, $n \in \mathbb{N}$ is in unary representation, c is a coloring of the upper and lower border of the $n \times n$ grid, and t is a D-tiling of the $n \times n$ grid that respects $c\}$.

Similar to SAT (Theorem of Cook [Coo71]) the NP-completeness of TILING was shown by simulating the work of a nondeterministic polynomial-time Turing machine on an arbitrary input using tiles [GJP77]. A published version of this result can be found in [vEB97]. Analogous to the case of SAT this reduction turns out to be a *gp*-reduction. Without going into details we want to describe the idea of that reduction.

Let A be an arbitrary NP-problem and let V_A be a verifier for A. Let M be the nondeterministic Turing machine that, given an input x, in a first nondeterministic phase guesses all possible potential solutions for x. Afterwards, M verifies for each guessed potential solution y (by using a deterministic polynomial time algorithm for V_A) whether (x, y) is in V_A or not. In case of $(x, y) \in V_A$ the corresponding path of the computation of $M(x)$ accepts and otherwise, it rejects. We assume that M is standardized in a way, such that each computational path of a computation $M(x)$ rejects or accepts after exactly $p(|x|)$ steps, where p is a polynomial. Furthermore, in case of $M(x)$ accepts, it finishes in a fixed accepting configuration.

Now, an input x for M is mapped to the triple $(D, p(|x|), c)$ of a specially chosen set of tile types D, the unary representation of $p(|x|)$, and a specially chosen coloring c. The main idea is, that each line of the grid square $p(|x|) \times p(|x|)$ represents a configuration of the computation of M. The set of tile types D is chosen in a way such that two consecutive lines fit together if and only if the upper line represents a configuration that is a valid successor (according to the program of M) of the configuration represented by the lower line. The coloring c is chosen, such that the bottom line represents the starting configuration of the computation of $M(x)$ and that the top line represents the accepting configuration of M. Then, it is not hard to see that each accepting path of the computation

$M(x)$ leads to a valid D-tiling of the square $p(|x|) \times p(|x|)$ and vice versa. Since each accepting path of the computation $M(x)$ represents a solution for x we have a one-to-one-correspondence between the solutions of x and the D-tilings of the square $p(|x|) \times p(|x|)$, that is, a reduction $(A, V_A) \leq^p_{gp} (\text{TILING}, V_{\text{TILING}})$.

Theorem 4.2.5. TILING *is $\exists_r \forall_l gp$-complete via V_{TILING}.*

4.2.4 Exact-3-Cover (X3C)

Our intention is to show $\exists_r \forall_l gp$-completeness for the six basic NP-complete problems (see [GJ79]). To show $\exists_r \forall_l gp$-completeness of the next candidate, 3DIMENSIONAL MATCHING (3DM), we will first take a closer look at the strongly related problem EXACT-3-COVER (X3C).

Problem Description (Exact-3-Cover)

Given: A pair $(V, \mathfrak{C})$, where V is a set such that $|V|$ is divisible by 3, and $\mathfrak{C}$ is a collection of 3-element subsets of V.

Question: Is there an exact-3-cover $\mathfrak{C}'$ of $(V, \mathfrak{C})$, i.e., a subcollection $\mathfrak{C}' \subseteq \mathfrak{C}$ such that each element of V appears in exactly one member of $\mathfrak{C}'$?

Standard Verifier:

$$V_{\text{X3C}} = \{((V, \mathfrak{C}), \mathfrak{C}') : |V| \text{ is devisible by 3, } \mathfrak{C} \text{ is a collection of 3-element subsets of } V, \text{ and } \mathfrak{C}' \subseteq \mathfrak{C} \text{ is an exact-3-cover of } (V, \mathfrak{C})\}.$$

Note that each X3C-instance $(V, \mathfrak{C})$ can be interpreted as a 3-hypergraph. A hypergraph is a pair (V, E), where V is a finite set of vertices and E is a collection of nonempty subsets of V, called hyperedges. A k-hypergraph is a hypergraph, where each hyperedge consists of exactly k vertices. So it is not hard to see that each proper X3C-instance $(V, \mathfrak{C})$ is also a 3-hypergraph. Following this interpretation, we will call the elements of V vertices and the elements of $\mathfrak{C}$ hyperedges. We will often use edge as a shorthand for hyperedge. Analogous to standard graphs, we will imagine an X3C-instance $(V, \mathfrak{C})$ as vertices and edges embedded in the plane.

Although $\exists_r \forall_l gp$-completeness of X3C has implicitly been shown in [HMRS98] by proving #P-completeness of X3C (the given parsimonious reduction from 3SAT can easily be extended to be generating parsimonious) we give an alternative proof for the $\exists_r \forall_l gp$-completeness of X3C that will be more helpful in showing the $\exists_r \forall_l gp$-completeness of 3DM. Furthermore, we feel that this proof is especially beautiful, since the used construction reflects the geometry of a tiling very intuitively.

Theorem 4.2.6. X3C *is $\exists_r \forall_l gp$-complete via V_{X3C}.*

Proof. It suffices to show that $(\text{TILING}, V_{\text{TILING}}) \leq^p_{gp} (\text{X3C}, V_{\text{X3C}})$ via functions f and g (Lemma 4.1.10, Theorem 4.2.5).

Let (D, n, c) be an instance for TILING. We denote the number of tile types $|D|$ by m and we denote the elements of D by $d_1, d_2, \ldots, d_m$. Note that deciding containment of (D, n, c) in TILING is trivial for $|D| = 0$ and $|D| = 1$. Hence, such instances (D, n, c) can trivially be mapped to proper fixed X3C-instances. Assume $|D| \geq 2$ in the following.

Recall that $(D, n, c) \in$ TILING if and only if there exists a D-tiling of the square $(1, \ldots, n) \times (1, \ldots, n)$ that extends the coloring c. The idea of our reduction is to simulate D-tilings of the $n \times n$-grid by creating an $n \times n$-grid of X3C-gadgets $\hat{G}$ (see Figure 4.4). For this gadget $\hat{G}$, there will be exactly m different possible exact-3-covers, each representing one tile-type from D. So an exact-3-cover of all n^2 gadgets will represent putting certain tile-types to all grid-positions.

Furthermore, the construction will achieve that the tile-types represented by the exact-3-covers of gadgets in adjacent grid positions fit together and that the tile-types associated to the exact-3-covers of gadgets at the upper and lower border are colored in accordance to the given coloring c. Assembling all these parts, it will not be hard to see that there is a bijection between the possible D-tilings of the square $(1, \ldots, n) \times (1, \ldots, n)$ respecting c and the possible covers of the constructed X3C-instance $(V, \mathfrak{C})$ (see also Figure 4.4).

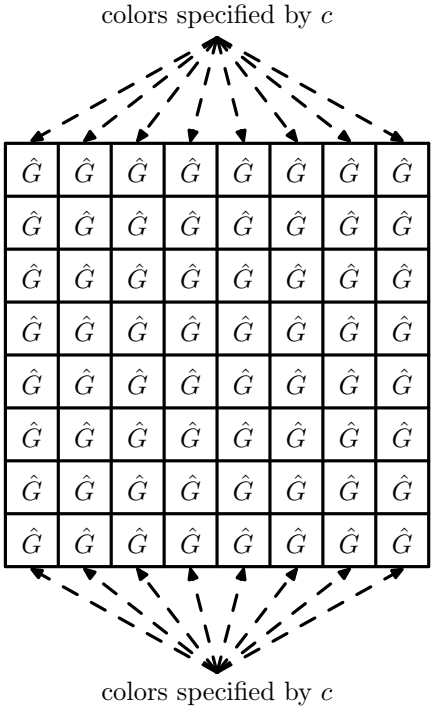

Figure 4.4: A sketch of the X3C-instance we will construct, for $n = 8$.

The first step of explicitly explaining the reduction is defining the above men-

tioned X3C-gadget $\hat{G}$. It consists of m copies $G_1, \ldots, G_m$ of a smaller gadget G. The copy G_i represents the tile type d_i, $1 \leq i \leq m$. Each gadget G_i consists of the vertices n_i, e_i, s_i, w_i (for north, east, south, and west), of three internal auxiliary vertices x_i, y_i, and z_i and of the edges $\{n_i, x_i, w_i\}$,$\{e_i, s_i, y_i\}$, and $\{x_i, y_i, z_i\}$ (see Figure 4.5). Note that the two internal vertices x_i and y_i will not appear in any other edge of the to be constructed X3C-instance. Hence, we can state the following lemma.

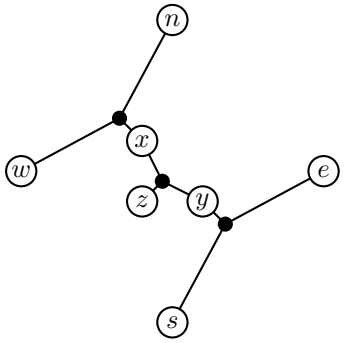

Figure 4.5: A depiction of the gadget G, whose copy G_i represents the tile type d_i, $1 \leq i \leq m$. Each filled dot represents an edge, namely the one that includes the three connected vertices.

Lemma 4.2.7. *Let the gadget G (see Figure 4.5) be included in an* X3C*-instance $(V, \mathfrak{C})$ such that the vertices x and y do not appear in any other edge from $\mathfrak{C}$. Then for each exact cover $\mathfrak{C}'$ for $(V, \mathfrak{C})$ either holds*

- $\mathfrak{C}' \cap \{\{x, y, z\}, \{n, x, w\}, \{e, s, y\}\} = \{\{x, y, z\}\}$ *or*
- $\mathfrak{C}' \cap \{\{x, y, z\}, \{n, x, w\}, \{e, s, y\}\} = \{\{n, x, w\}, \{e, s, y\}\}$.

Proof. The proof is a very simple distinction of cases and thus, omitted. □

We will call an exact-3-cover $\mathfrak{C}'$ of $(V, \mathfrak{C})$ an 1-cover of G if and only if $\mathfrak{C}'$ includes $\{x, y, z\}$ and we will call it 0-cover if and only if $\mathfrak{C}'$ includes the edges $\{n, x, w\}$ and $\{e, s, y\}$. Note that in a 1-cover, the vertices n, e, s, and w are not covered by one of edges of the gadget G and in the 0-cover the vertex z is not covered by any edge of the gadget. These vertices must be covered by some alternative edges. Following this, we will call a gadget-vertex covered internally if it is covered by an edge belonging to the gadget and we will call it covered externally, otherwise.

Now, we return to the construction of the gadget $\hat{G}$. As mentioned in the beginning $\hat{G}$ consists of m copies $G_1, \ldots, G_m$ of G. The construction shall make sure that each exact-3-cover of $\hat{G}$ 1-covers exactly one of the copies, say G_i, and 0-covers the remaining $n-1$ copies. Then we will say that this situation corresponds to having the tile-type d_i in this grid position.

In order to achieve this, we introduce another type of gadget (named H) that will ensure that exactly one copy G_i is 1-covered. This connection between the gadgets $G_1, \ldots, G_m$ will be realized using the auxiliary vertices z_i.

The gadget H is defined as follows. It consists of the vertices $z_1, z_2, \ldots, z_m$, $h_1, h_2, \ldots, h_{m-1}$, and $h'_1, h'_2, \ldots, h'_{m-1}$ and of the edges $\{z_i, h_i, h'_i\}, 1 \leq i \leq m-1$, and $\{z_{i+1}, h_i, h'_i\}, 1 \leq i \leq m-1$ (see Figure 4.6).

Lemma 4.2.8. *Let the gadget H (see Figure 4.6) be included in an* X3C*-instance $(V, \mathfrak{C})$ such that the vertices $h_i, h'_i, 1 \leq i \leq m-1$, do not appear in any other edge from $\mathfrak{C}$. If $\mathfrak{C}'$ is an exact-3-cover for $(V, \mathfrak{C})$ then the auxiliary vertices $h_i, h'_i, 1 \leq i \leq m-1$, are covered according to one of m different possibilities. For each i, $1 \leq i \leq m$, there is exactly one of these possibilities such that $z_1 \ldots, z_{i-1}, z_{i+1}, \ldots, z_m$ are covered internally and z_i is covered by an external edge.*

Proof. Let H be included in $(V, \mathfrak{C})$ in the described way and let $\mathfrak{C}'$ be an exact-3-cover for $(V, \mathfrak{C})$. For each natural number $i, 1 \leq i \leq m-1$, the auxiliary vertices h_i and h'_i from H are covered by one of the edges $\{z_i, h_i, h'_i\}$ and $\{z_{i+1}, h_i, h'_i\}$. Obviously, $\{z_{i+1}, h_i, h'_i\} \in \mathfrak{C}'$ implies $\{z_{i+2}, h_{i+1}, h'_{i+1}\} \in \mathfrak{C}'$ and thus $\{z_{i+3}, h_{i+2}, h'_{i+2}\} \in \mathfrak{C}'$ and so on. Analogously, $\{z_i, h_i, h'_j i\} \in \mathfrak{C}'$ implies $\{z_{i-1}, h_{i-1}, h'_{i-1}\} \in \mathfrak{C}'$ and thus $\{z_{i-2}, h_{i-2}, h'_{i-2}\} \in \mathfrak{C}'$ and so on. Hence, there exists a natural number k, $1 \leq k \leq m$, such that $\{z_i, h_i, h'_i\} \in \mathfrak{C}', 1 \leq i \leq k-1$ and $\{z_{j+1}, h_j, h'_j\} \in \mathfrak{C}', k \leq j \leq n-1$, that is, all H-vertices but z_k are covered internally. Since there is exactly one such covering for each k, $1 \leq k \leq m$, the lemma is proven. □

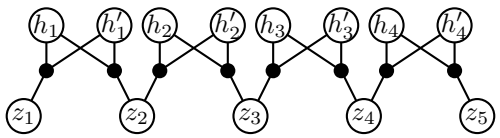

Figure 4.6: The gadget H, for $m = 5$.

With the help of the gadget H we can now connect the gadgets $G_1, \ldots, G_m$ by identifying each z_i from G_i with the vertex z_i in H. That completes the construction of $\hat{G}$. Now, exactly one vertex $z_i \in H$ must be covered externally (Lemma 4.2.8). Hence, while covering $G_1, \ldots, G_m$, z_i is covered internally and $z_1, \ldots, z_{i-1}, z_{i+1}, \ldots, z_m$ are covered externally. It follows that all gadgets $G_1, \ldots, G_{i-1}, G_{i+1}, \ldots, G_m$ are 0-covered and G_i is 1-covered.

The internal vertices of all gadgets involved in $\hat{G}$ (x_i and y_i in the gadgets G_i and all the vertices h_i and h'_i in H) will not appear in any of the edges of $(V, \mathfrak{C})$, besides the ones belonging to the gadgets. Hence, we obtain the following Lemma.

Lemma 4.2.9. *Let the structure $\hat{G}$ be included in an* X3C*-instance $(V, \mathfrak{C})$ such that the internal vertices of all involved gadgets $G_i, 1 \leq i \leq m$ and H (x_i, y_i, and z_i in the gadgets $G_i, 1 \leq i \leq m$, and all the vertices h_j and $h'_j, 1 \leq j \leq m-1$,*

in H) do not appear in any other edge from $\mathfrak{C}$. Then for each exact cover $\mathfrak{C}'$ for $(V, \mathfrak{C})$ the following holds:

1. *Exactly one of the gadgets $G_1, \ldots, G_m$ is 1-covered, say G_j.*
2. *Almost all vertices of the gadget $\hat{G}$ are covered internally (by edges that belong to one of the involved gadgets G_i or H). Only the four vertices n_j, e_j, s_j, and w_j from the gadget G_j are covered by edges not belonging to $\hat{G}$.*
3. *The internal covering is unique, that is, if $\mathfrak{C}'$ covers the vertices n_i, e_i, s_i, and w_i for some $i, 1 \leq i \leq m$, externally (G_i is 1-covered) then $\mathfrak{C}'$ covers the rest of $\hat{G}$ in a predetermined way.*

Proof. The Lemma follows from Lemma 4.2.7 and Lemma 4.2.8. □

Now, we put a copy of the gadget $\hat{G}$ in each of the n^2 grid positions and name the copy in the xth line and yth column *grid position* $GP_{(x,y)}$. To avoid complicated indices we do not include the grid position into the notation of the gadgets $G_1, \ldots, G_m$, and H in $\hat{G}$ and their vertices. When talking about one of those gadgets or vertices it will be clear from the context in which grid position they are. We have seen that in each grid position all but four vertices (n_i, e_i, s_i, w_i for some $i, 1 \leq i \leq m$) can and must be covered internally. Thereby, covering all vertices but n_i, e_i, s_i, w_i internally stands for putting the tile type d_i at this grid position. So what we still have to manage is the issue that adjacent tiles shall fit together.

The basic idea is the same for each pair of adjacent grid positions, so we only describe how we connect the gadgets $GP_{(1,1)}$ and $GP_{(1,2)}$ in the grid positions $(1, 1)$ and $(1, 2)$. First, we place a new auxiliary vertex s on the common border of both positions. Recall, that one of the vertices $e_i, 1 \leq i \leq m$, in $GP_{(1,1)}$ and one of the vertices $w_j, 1 \leq j \leq m$, in $GP_{(1,2)}$ has to be covered externally, representing having tile-type d_i in grid position $(1, 1)$ and d_j in $(1, 2)$. In order to ensure that the color of the eastern triangle of d_i equals the color of the western triangle of d_j we only add the edges $\{\{e_i, s, w_j\} : east(d_i) = west(d_j), 1 \leq i, j \leq m\}$ to $\mathfrak{C}$. Note that this will be the only external edges that will include, and thus will be able to cover, the vertices e_i from $GP_{(1,1)}$ and w_j from $GP_{(1,2)}$. Thus, we can state the following lemma.

Lemma 4.2.10. *Let the structures $GP_{(1,1)}$ and $GP_{(1,2)}$ be connected as mentioned above in an* X3C*-instance $(V, \mathfrak{C})$ such that the internal vertices of all involved gadgets G_i and H (x_i and y_i in the gadgets $G_i, 1 \leq i \leq m$, and all the vertices h_j and $h'_j, 1 \leq j \leq m-1$, in H) and the vertex s connecting $GP_{(1,1)}$ and $GP_{(1,2)}$ do not appear in any other edge from $\mathfrak{C}$. Then for each exact-3-cover $\mathfrak{C}'$ for $(V, \mathfrak{C})$ the tiles, associated to the covering of $GP_{(1,1)}$ and $GP_{(1,2)}$, fit together. Furthermore, the internal cover of the structures $GP_{(1,1)}$ and $GP_{(1,2)}$ uniquely determines the cover of the auxiliary vertex s (and hence the cover of the vertices e_i in $GP_{(1,1)}$ and w_j in $GP_{(1,2)}$).*

Proof. Let $\mathfrak{C}'$ be an exact cover for $(V, \mathfrak{C})$. Let d_i (d_j) be the tile-type associated to the cover of $GP_{(1,1)}$ ($GP_{(1,2)}$). Hence, e_i in $GP_{(1,1)}$ and w_j in $GP_{(1,2)}$ are covered externally. The only external edge that might be able to cover these vertices (and the auxiliary vertex s) is $\{e_i, s, w_j\}$. Hence, $\{e_i, s, w_j\}$ is included in $\mathfrak{C}$, implying that $\{e_i, s, w_j\} \in \{\{e_i, s, w_j\} : east(d_i) = west(d_j), 1 \leq i, j \leq m\}$. Thus, $east(d_i) = west(d_j)$, that is, the tiles d_i and d_j fit together. □

Analogously, we connect all other pairs of adjacent structures and thus, we have an analogon to Lemma 4.2.10 for all pairs of adjacent grid positions.

The only issues we still have to take care of are the structures at the border of the grid, for instance the structure $GP_{(1,1)}$. Recall that by Lemma 4.2.9 we have that for each covering of $GP_{(1,1)}$ exactly one of the vertex sets $\{n_j, e_j, s_j, w_j\}, 1 \leq j \leq m$, must be covered externally. Note that the vertices e_j and s_j can (and must) be covered by one of the edges connecting $GP_{(1,1)}$ with $GP_{(1,2)}$ and $GP_{(1,1)}$ with $GP_{(2,1)}$, respectively. But so far, there is no possibility to cover the vertices n_j and w_j.

First, we will consider the northern part of $GP_{(1,1)}$. Note that we must ensure that the color of the northern triangle of the tile type in $(1, 1)$ equals the color $c_{n(1,1)}$ predetermined by the coloring c of the upper and lower border of the grid. To do so, we add new vertices n', n'' and the edges $\{\{n', n'', n_k\} : north(d_k) = c_{n(1,1)} \wedge 1 \leq k \leq m\}$. Now, let $\mathfrak{C}'$ be an exact cover of $(V, \mathfrak{C})$ that represents placing the tile d_j in $(1, 1)$, so n_j in $GP_{(1,1)}$ is covered externally. Since the edges $\{\{n', n'', n_k\} : north(d_k) = c_{n(1,1)}$ and $1 \leq k \leq m\}$ are the only possibilities to cover n_j, it follows that d_j has a northern triangle with the color $c_{n(1,1)}$. Conversely, it is easy to see that assuming the tile d_j in $(1, 1)$ has the correct northern color, the vertex n_j can be covered by exactly one of the added edges.

For the western part of $GP_{(1,1)}$ we apply almost the same construction. The only difference is that we can omit the constraint on the color, since the color on the left and right border is not predetermined by c. So we add auxiliary vertices w' and w'' and the edges $\{w', w'', w_k\}, 1 \leq k \leq m$. Obviously, no matter which border vertex $w_j, 1 \leq j \leq m$, has to be covered, it can be done by exactly one of the added edges.

We transfer the first construction to each grid position on the upper and lower border of the $n \times n$ square and the second construction to the grid positions on the right and left border. Note that in doing so we always use new pairs of auxiliary vertices n', n'' and w', w'', respectively[2].

Let us take stock. Given a TILING-instance (D, n, c), the constructed X3C-instance $(V, \mathfrak{C})$ consists of an $n \times n$ grid of the structures $GP_{(i,j)}, 1 \leq i, j \leq n$. Each pair of adjacent grid structures, for instance $GP_{(1,1)}$ and $GP_{(1,2)}$, is connected as described before Lemma 4.2.10 by an auxiliary vertex and edges. And finally, as just described, at each of the $4n$ border segments of the $n \times n$ grid, there

[2]One may wonder about the repeated names of these vertices, but recall that we abstain from indexing these vertices with their grid positions. Formally, all vertices have such an index indicating the grid position.

are two auxiliary vertices and some edges. We define the function f to map the TILING-instance (D, n, c) to the constructed X3C-instance $(V, \mathfrak{C})$. It is easy to see that f is polynomial-time computable.

Since we want to give a *gp*-reduction we further need to specify a polynomial-time computable bijection g between the solutions of (D, n, c) and the solutions of $f((D, n, c))$. To see how to define g let t be a D-tiling for (D, n, c) respecting c. We gain an exact cover $\mathfrak{C}'_t$ of $(V, \mathfrak{C})$ as follows. We cover each grid structure $GP_{(i,j)}$ in the way that is associated with using the tile type d_k, where $k = t(i, j)$, in the sense of Lemma 4.2.9. Since the tile types assigned to adjacent square positions fit together, Lemma 4.2.10 provides a unique way to cover the auxiliary vertices between adjacent grid positions. So we cover them this way. A closer look shows that all but the $4n$ border vertices and the associated pairs of auxiliary vertices are already covered. Since t is respecting the colors given by c on the border, we can also cover all these vertices. So we have an exact cover $\mathfrak{C}'_t$ of $(V, \mathfrak{C})$ and we define the generating function g by $g((D, n, c), t) := \mathfrak{C}'_t$. Obviously, g is polynomial-time computable and injective.

To show that g is surjective, and hence bijective, let $\mathfrak{C}'$ be an exact cover of the X3C-instance $f((D, n, c)) = (V, \mathfrak{C})$. Lemma 4.2.9 ensures that each structure $GP_{(i,j)}, 1 \leq i, j \leq n$, is covered according to one of m possibilities, each representing a certain tile type from D. Moreover, by Lemma 4.2.10 we have that the tiles in adjacent grid positions fit together and that the auxiliary vertices between adjacent grid positions are covered in a unique way. It is easy to see that the cover of the border vertices is also uniquely defined by the tiles chosen in the involved grid positions. Hence, the way the structures $GP_{(i,j)}, 1 \leq i, j \leq n$, are covered, uniquely defines the remainder of the exact cover of $(V, \mathfrak{C})$. Furthermore it holds that such a covering stands for a D-tiling $t_{\mathfrak{C}'}$ and it is easy to see that $\mathfrak{C}' = \mathfrak{C}'_{t_{\mathfrak{C}'}} = g((D, n, c), t_{\mathfrak{C}'})$. Thus, $\mathfrak{C}'$ is part of the range of g and we have that g is bijective.

Hence, we have a *gp*-reduction $(\text{X3C}, V_{\text{X3C}}) \leq^p_{gp} (\text{TILING}, V_{\text{TILING}})$ via f and g, and thus $\exists_r \forall_l gp$-completeness of X3C via V_{X3C}. □

4.2.5 3Dimensional Matching (3DM)

In this section we benefit from our efforts made in Section 4.2.4 by modifying the proof of Theorem 4.2.6 to also work for 3DIMENSIONAL MATCHING.

3Dimensional Matching (3DM)

Problem Description (3Dimensional Matching)

Given: A 4-tuple (S, X, Y, Z), where X, Y, and Z are sets having the same number q of elements and S is a subset of $X \times Y \times Z$.

Question: Is there a 3D-matching for S, i.e., is there a subset $M \subseteq S$ such that $|M| = q$ and no two triples from M agree in any coordinate ?

Standard Verifier:

$V_{\text{3DM}} = \{((S, X, Y, Z), M) : X, Y \text{and } Z \text{ are sets having the same number } q$ of elements ($q = |X| = |Y| = |Z|$), S is a subset of $X \times Y \times Z$, and M is a 3D-matching for $S\}$.

The following lemma is the tool which will allow us to modify the proof of Theorem 4.2.6 to also work for 3DIMENSIONAL MATCHING. To state the lemma we need the notion of 3-colorings of X3C-instances: A mapping $c : V \to \{1, 2, 3\}$ for a given X3C-instance $(V, \mathfrak{C})$ is called a 3-*coloring* for $(V, \mathfrak{C})$ if and only if each edge $\{v_1, v_2, v_3\}$ from $\mathfrak{C}$ is 3-colored, that is, $\{c(v_1), c(v_2), c(v_3)\} = \{1, 2, 3\}$. Note that we still interpret X3C-instances $(V, \mathfrak{C})$ as hypergraphs.

Lemma 4.2.11. *Let $(V, \mathfrak{C})$ be an* X3C*-instance having a 3-coloring c. Furthermore, let*

$$X_c := \{v : v \in V \wedge c(v) = 1\},$$
$$Y_c := \{v : v \in V \wedge c(v) = 2\},$$
$$Z_c := \{v : v \in V \wedge c(v) = 3\}, \text{ and}$$
$$S_c := \{(x, y, z) : \{x, y, z\} \in \mathfrak{C} \wedge c(x) = 1 \wedge c(y) = 2 \wedge c(z) = 3\} \subseteq X_c \times Y_c \times Z_c.$$

Then $(V, \mathfrak{C})$ is in X3C *if and only if (S_c, X_c, Y_c, Z_c) is in* 3DM *and there exists a (canonical) bijective mapping between the solution sets $V_{\text{X3C}}((X, \mathfrak{C}))$ and $V_{\text{3DM}}((S_c, X_c, Y_c, Z_c))$.*

Proof. Let $(V, \mathfrak{C})$ be an X3C-instance, let c be a 3-coloring of $(V, \mathfrak{C})$ and let (S_c, X_c, Y_c, Z_c) be the 3DM-instance constructed from $(V, \mathfrak{C})$ and c as described above.

Assume that $\mathfrak{C}'$ is an exact cover for $(V, \mathfrak{C})$. It follows that $\mathfrak{C}' = |V|/3$ and since each edge from $\mathfrak{C}'$ contains one element of each color, it also follows that $q := |X_c| = |Y_c| = |Z_c| = |V|/3$. We convert the set $\mathfrak{C}'$ of 3-colored edges to a set $M_{\mathfrak{C}'}$ of triples such that the edge $\{v_1, v_2, v_3\}$ with $c(v_i) = i$, $1 \leq i \leq 3$, is converted to the triple (v_1, v_2, v_3). By the definition of S_c we have that $M_{\mathfrak{C}'}$ is a subset of S_c. Obviously, we have $|M_{\mathfrak{C}'}| = |\mathfrak{C}'| = q$ and since $\mathfrak{C}'$ exactly covers V, it follows that no two triples from $M_{\mathfrak{C}'}$ agree in any coordinate. Thus $M_{\mathfrak{C}'}$ is a 3D-matching of S_c, i.e., a solution for (S_c, X_c, Y_c, Z_c).

Conversely, assume that M is a solution of the constructed 3DM-instance (S_c, X_c, Y_c, Z_c). Since $|M|$ equals $|X_c|, |Y_c|$, and $|Z_c|$ and no triples from M agree in any coordinate, we have that each element from X_c appears in exactly one triple from M at the first coordinate. Analogously, each element from Y_c and Z_c appears in exactly one triple from M in the second and the third coordinate, respectively. It is a simple implication that the set $\mathfrak{C}'_M$ of edges arising from M by converting triples to edges, is an exact cover of V and a subset of $\mathfrak{C}$. Thus, $\mathfrak{C}'_M$ is a solution for $(V, \mathfrak{C})$. Obviously, after converting $\mathfrak{C}'$ to $M_{\mathfrak{C}'}$ and converting $M_{\mathfrak{C}'}$ to $\mathfrak{C}'_{M_{\mathfrak{C}'}}$, it holds that $\mathfrak{C}'_{M_{\mathfrak{C}'}} = \mathfrak{C}'$. Thus, the mapping $\mathfrak{C}' \mapsto M_{\mathfrak{C}'}$ is a bijection between the solution sets $V_{\text{X3C}}((V, \mathfrak{C}))$ and $V_{\text{3DM}}((S_c, X_c, Y_c, Z_c))$. □

Theorem 4.2.12. 3DM *is $\exists_r \forall_l gp$-complete via V_{3DM}.*

Proof. The idea of the proof is to modify the reduction $(\text{Tiling}, V_{\text{Tiling}}) \leq^p_{gp} (\text{X3C}, V_{\text{X3C}})$ (via f and g) given in the proof of Theorem 4.2.6 to work for 3DM by making clever use of Lemma 4.2.11. The mentioned gp-reduction maps an instance (D, n, c') for Tiling to an instance $f((D, n, c')) = (V, \mathfrak{C})$ for X3C such that there exists a bijection g between their solution sets. If we manage to give a 3-coloring c for $(V, \mathfrak{C})$, by Lemma 4.2.11 we gain an instance $(S_c, X_c, Y_c, Z_c) =: f'((V, \mathfrak{C}))$ for 3DM and a canonical bijection g' between the solution sets $V_{\text{X3C}}((V, \mathfrak{C}))$ and $V_{\text{3DM}}((S_c, X_c, Y_c, Z_c))$. So, it is not hard to see that by combining the reduction functions f and g and the mappings f' and g' we gain a gp-reduction $(\text{Tiling}, V_{\text{Tiling}}) \leq^p_{gp} (\text{3DM}, V_{\text{3DM}})$ via $f \circ f'$ and $g \circ g'$.

Hence, it is sufficient to show that all X3C-instances produced by the reduction function f, formally defined in the proof of Theorem 4.2.6, are 3-colorable. To do so, let (D, n, c') be an instance for Tiling. Recall that D is the set of tile types that shall be used to tile the $n \times n$ grid respecting the colors given by c' on the upper and lower border of the grid. Now, let $f((D, n, c')) = (V, \mathfrak{C})$. We will explicitly give a 3-coloring g of $(V, \mathfrak{C})$, that is, a mapping $c : V \to \{1, 2, 3\}$ such that each edge from $\mathfrak{C}$ contains vertices from all three colors.

First, we will have a look at the most simple gadget used in $(V, \mathfrak{C})$, which is the gadget G. For each copy of the gadget G we use one of the two colorings given in Figure 4.7.

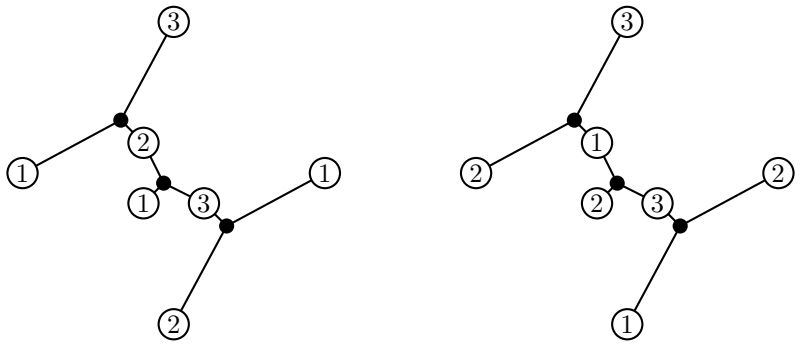

Figure 4.7: Two possible colorings of the gadget G. For clarity of the figure we replaced the labels of the vertices (see Figure 4.5) by the used color.

We color all $|D| =: m$ versions $G_1, \ldots, G_m$ of G in a position (i, j) in one of the two ways depicted in Figure 4.7. The choice of the coloring depends on the grid position - all grid positions in odd columns are covered as shown in the left of Figure 4.7 and grid positions in even columns are colored as shown in the right. For clarity, the respective choice of the coloring, depending on the grid position, is illustrated in Figure 4.8 for a small example.

In each grid position depending on the above choice of the coloring, the gadget H is colored in one of the two ways given in Figure 4.9 such that the color of the

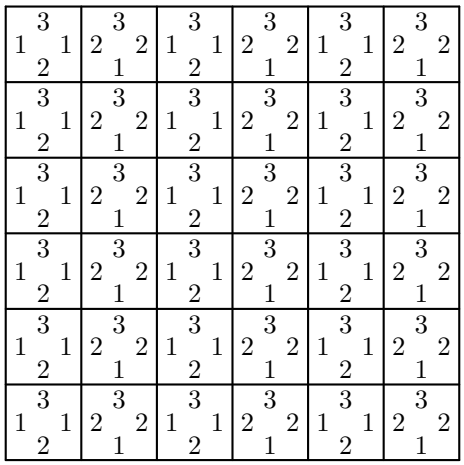

Figure 4.8: Illustration of the coloring of the gadgets $G_1, \ldots, G_m$, depending on their grid position. Disregarding the inner coloring we only give the colors of the copies of the m tuples (n_i, e_i, s_i, w_i), $1 \leq i \leq m$. In each square, the northern number represents the color of the n_i, the eastern number represents the color of e_i and so on.

common vertices $z_i, 1 \leq i \leq m$, is the same as in the chosen coloring of $G_1 \ldots, G_m$. So if the versions of G are colored as shown in the left (right) part of Figure 4.7, the corresponding gadget H is colored as in the upper (lower) image of Figure 4.9.

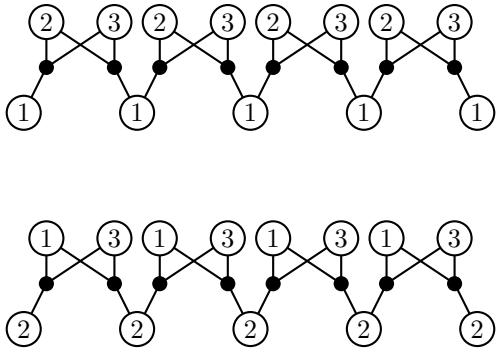

Figure 4.9: Two valid colorings of the gadgets H illustrated for $m = 5$. Again we replaced the labelling of the vertices (see Figure 4.6) by the used colors for clarity.

Thus, we have the choice between two valid colorings for the gadget $\hat{G}$ in each grid position (i, j). Note that the only vertices of each $GP_{(i,j)}$ that are included in edges outside of $GP_{(i,j)}$, are the vertices n_i, e_i, s_i, and w_i in the m copies $G_1 \ldots, G_m$ of G. Disregarding the inner coloring of a $GP_{(i,j)}$ we have two possible colorings, one assigns the colors $(3, 1, 2, 1)$ to all tuples of vertices (n_i, e_i, s_i, w_i)[3],

[3]Naturally, this notion stands for assigning 3 to all n_i, 1 to all e_i and so on.

$1 \leq i \leq m$, and another one that assigns $(3, 2, 1, 2)$.

The only connection between adjacent grid positions are the auxiliary vertices between them and the associated edges, whose purpose was to ensure that adjacent tiles fit together. Figure 4.8 shows that for instance, all e_i in $GP_{(1,1)}$ are colored with 1 and all w_i in $GP_{(1,2)}$ are colored with 2 and thus, we can assign 3 to the auxiliary vertex between $GP_{(1,1)}$ and $GP_{(1,2)}$. Furthermore, it is not hard to see, that this can also be done for all other pairs of adjacent grid positions.

The only remaining uncolored vertices are the auxiliary vertices on the edge of the grid. Since the argumentation will obviously work the same for all pairs of auxiliary vertices, it suffices to have a closer look at the two auxiliary vertices n', n'', associated to the northern vertices n_i in $GP_{(1,1)}$. Recall that the only edges that include n' and n'' are of the type $\{n', n'', n_i\}, 1 \leq i \leq m$. Since all vertices n_i in $GP_{(1,1)}$ are colored by 3 (see Figure 4.8), we gain a valid coloring by assigning the remaining colors 1 to n' and 2 to n''.

Overall, we have managed to give a 3-coloring of the X3C-instance $(V, \mathfrak{C})$. As we have seen in the beginning of the proof, we obtain a *gp*-reduction from $(\text{TILING}, V_{\text{TILING}})$ to $(\text{3DM}, V_{\text{3DM}})$ and thus the theorem. □

4.2.6 Partition (Part)

When we dealt with PARTITION the first time in Section 3.3.3 we gave a not so common problem definition using multisets. We did so in order to be able to prove the NP-completeness of all n + PARTITION by giving a *gp*-reduction 1 + PARTITION $\leq^p_{gp}$ PARTITION, a so called auto-reduction (see Section 3.3.3 for further explanations). In this section we give an inter-reduction 3DM $\leq^p_{gp}$ PARTITION, which implies the $\exists_r\forall_l$*gp*-completeness of PARTITION. In doing so, we can use the following more common problem definition.

Problem Description (Partition)

Given: A finite set A and a mapping s from A to $\mathbb{N}$.

Question: Is there a partition of A, that is, a subset $A' \subseteq A$ such that

$$\sum_{a \in A'} s(a) = \sum_{a \notin A'} s(a) \ ?$$

Standard Verifier:

$$V_{\text{PART}} = \{((A, s), A') : A \text{ is a finite set, } s \text{ is a mapping from } A \text{ to } \mathbb{N}, \text{ and } A' \text{ is a subset of } A \text{ such that } \sum_{a \in A'} s(a) = \sum_{a \notin A'} s(a)\}.$$

The NP-completeness of PARTITION can be shown via a many-one reduction from 3DM (see [GJ79]). It is easy to see that this reduction is even parsimonious and

can easily be made generating parsimonious. Thus, we have $(3\text{DM}, V_{3\text{DM}}) \leq^p_{gp}$ (PARTITION, V_{PART}). Since 3DM is $\exists_r \forall_l gp$-complete (Theorem 4.2.12), the $\exists_r \forall_l gp$-completeness of PARTITION follows immediately.

Theorem 4.2.13. PARTITION *is* $\exists_r \forall_l gp$*-complete via* V_{PART}.

4.2.7 Vertex Cover (VC)

Problem Description (Vertex Cover)

Given: A graph $G = (V, E)$ and a natural number k.

Question: Is there a vertex cover of size at most k for G, i.e., is there a vertex set $V' \subseteq V$ such that each edge from E is incident to a vertex from V' ?

Standard Verifier:

$V_{\text{VC}} = \{((G, k), V') : G = (V, E)$ is a graph, k is a natural number, and V' is a vertex cover of size at most k for $G\}$.

Theorem 4.2.14. VC *is* $\exists_r \forall_l gp$*-complete via* V_{VC}.

Proof. Since X3C is $\exists_r \forall_l gp$-complete via V_{X3C}, it suffices to give a gp-reduction from $(\text{X3C}, V_{\text{X3C}})$ to $(\text{VC}, V_{\text{VC}})$ via functions f and g. Note that we do not interpret X3C-instances as hypergraphs anymore. So let $(X, \mathfrak{C})$ be an instance for X3C such that $X = \{x_1, x_2, \ldots, x_l\}$ and $\mathfrak{C}$ is a collection of three-element subsets of X. If ℓ does not equal $3q$ for some natural number q, $(X, \mathfrak{C})$ has no solutions and we define $f((X, \mathfrak{C}))$ to be a fixed VC-instance without solutions. So from now on, assume that $X = \{x_1, x_2, \ldots, x_{3q}\}$ for some $q \in \mathbb{N}$. Furthermore, it is obvious that $(X, \mathfrak{C})$ is not in X3C if X includes a variable that does not appear in any of the three-element sets in $\mathfrak{C}$. In this case we also map $(X, \mathfrak{C})$ to a VC-instance without solutions. So we also assume that each variable in X appears in at least one three-element set of $\mathfrak{C}$.

The instance $(X, \mathfrak{C})$ will be mapped to an instance (G, k) for VC, which will be described in the following. First, for each variable x_i from X we count the number of three-element sets in $\mathfrak{C}$ that include x_i and denote this number by $\#x_i$. Then, we create a $\#x_i$-clique C_{x_i} for each variable x_i from X, we furthermore add one vertex for each three-element set in $\mathfrak{C}$, and call these vertices $\mathfrak{C}$-vertices. Finally, we connect the $\mathfrak{C}$-vertex corresponding to the three-element set $\{x_i, x_j, x_k\}$ to one vertex in C_{x_i}, to one vertex in C_{x_j}, and to one vertex in C_{x_k}, where we ensure that no vertex in any clique C_{x_i} is connected to more than one $\mathfrak{C}$-vertex. Since the size of the clique C_{x_i} equals the number of appearances of x_i in three-element sets from $\mathfrak{C}$ each vertex from C_{x_i} is connected to exactly one $\mathfrak{C}$-vertex. This concludes the construction of the graph G (see Figure 4.10 for a small example) and we define the reduction function f as $f((X, \mathfrak{C})) = (G, 3|\mathfrak{C}| - 2q)$. The number $3|\mathfrak{C}| - 2q$

derives from the idea that each vertex cover will include exactly $\#x_i - 1$ vertices from C_{x_i}, for each $1 \leq i \leq 3q$, and q $\mathfrak{C}$-vertices, which sums up to $3|\mathfrak{C}| - 2q$. Obviously, f is polynomial-time computable.

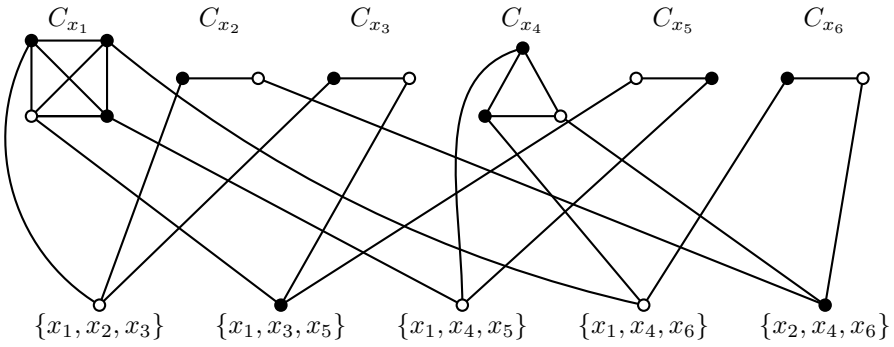

Figure 4.10: The constructed graph for the X3C-instance $(\{x_1, x_2, x_3, x_4, x_5, x_6\}, \{\{x_1, x_2, x_3\}, \{x_1, x_3, x_5\}, \{x_1, x_4, x_5\}, \{x_1, x_4, x_6\}, \{x_2, x_4, x_6\}\})$. The filled vertices form a vertex cover of the constructed graph corresponding to the X3C-solution $\{\{x_1, x_3, x_5\}, \{x_2, x_4, x_6\}\}$.

To complete the *gp*-reduction we further need to give a generating function, that is, a polynomial-time computable bijection g between the solutions of $(X, \mathfrak{C})$ and the solutions of $f((X, \mathfrak{C}))$. So let $\mathfrak{C}'$ be an exact-3-cover for $(X, \mathfrak{C})$. The generating function g maps $\mathfrak{C}'$ to the set V' that includes all $\mathfrak{C}$-vertices that correspond to a three-element set from $\mathfrak{C}'$ and that furthermore includes all vertices from all cliques C_{x_i} that are not connected to one of the chosen $\mathfrak{C}$-vertices (see also Figure 4.10 for an example). Obviously, g is polynomial-time computable. Note that since $\mathfrak{C}'$ is an exact-3-cover of $(X, \mathfrak{C})$, the chosen $\mathfrak{C}$-vertices are connected to exactly one vertex per clause C_{x_i}. So we have $|V'| = q+(\#x_1-1)+(\#x_2-1)+\cdots+(\#x_{3q}-1)$, since there are q $\mathfrak{C}$-vertices in V' and all but one of the $\#x_i$ vertices from each clique C_{x_i}, $1 \leq i \leq n$, are in V'. It is easy to see that $|V'| = q+(\#x_1-1)+(\#x_2-1)+\cdots+(\#x_{3q}-1) = \#x_1+\#x_2+\cdots+\#x_{3q}-2q = 3|\mathfrak{C}|-2q$, since the sum of all appearances of variables in $\mathfrak{C}$ is $3|\mathfrak{C}|$. To see that V' is actually a vertex cover of G let e be an edge of G. If e is an inner edge of some clique C_{x_i}, $1 \leq i \leq n$, then V' covers e because V' includes all but one of the vertices from C_{x_i}. If e is an edge between one of the cliques and one of the $\mathfrak{C}$-vertices we have to discuss two cases. If the incident $\mathfrak{C}$-vertex (say v) is a part of V', then V' obviously covers e. If v is not in V', then by the construction of V' the incident clique-vertex is in V' and thus, V' covers e. So V' is a vertex cover of proper size for G. It is immediate that different exact-3-covers of $(X, \mathfrak{C})$ are mapped to different vertex covers V', that is, g is injective.

We still have to show that g is also surjective. So let V' be a vertex cover of G of size at most $3|\mathfrak{C}| - 2q$. Observe that V' must contain at least $\#x_i - 1$ vertices of each clique $C_{x_i}, 1 \leq i \leq 3q$, because there is an edge between each pair of vertices in C_{x_i} that has to be covered. Since $(\#x_1 - 1) + \cdots + (\#x_{3q} - 1) =$

$\#x_1 + \cdots + \#x_{3q} - 3q = 3|\mathfrak{C}| - 3q$, V' contains at most q $\mathfrak{C}$-vertices. For a proof by contradiction assume that V' contains less than q $\mathfrak{C}$-vertices, say V' includes $k < q$ $\mathfrak{C}$-vertices. It follows that exactly $3k$ of the $3|\mathfrak{C}|$ edges between the cliques and the $\mathfrak{C}$-vertices are covered by these k vertices. Each of the at most $3|\mathfrak{C}| - 2q - k$ clique-vertices covers exactly one edge between the cliques and the $\mathfrak{C}$-vertices. This sums up to at most $3|\mathfrak{C}| - 2q - k + 3k = 3|\mathfrak{C}| - 2q + 2k < 3|\mathfrak{C}|$ covered edges. Since not all of the $3|\mathfrak{C}|$ edges between the cliques and the $\mathfrak{C}$-vertices are covered, we have a contradiction.

It follows that V' includes exactly $3|\mathfrak{C}| - 3q$ clique-vertices and q $\mathfrak{C}$-vertices. Consequently, in each clique $C_{x_i}, 1 \leq i \leq 3q$, there is one vertex that is not included in V'. Hence, the $3q$ edges between these $3q$ vertices and the $\mathfrak{C}$-vertices must be covered by the q $\mathfrak{C}$-vertices in V'. We denote the set of q three-element sets that correspond those q $\mathfrak{C}$-vertices by $\mathfrak{C}_{V'}$.

Since each $\mathfrak{C}$-vertex represents a three-element set $\{x_i, x_j, x_k\}$ and is connected to the three components C_{x_i}, C_{x_j}, and C_{x_k}, it follows that each variable x_i is included in exactly one of the three-element sets in $\mathfrak{C}_{V'}$, that is, $\mathfrak{C}_{V'}$ is an exact-3-cover of $(X, \mathfrak{C})$. Furthermore, it is not hard to see that g maps $\mathfrak{C}_{V'}$ to V', which reveals that g is surjective.

So $(\text{X3C}, V_{\text{X3C}})$ is gp-reducible to $(\text{VC}, V_{\text{VC}})$ via f and g, and thus, VC is $\exists_r \forall_l gp$-complete via V_{VC}. □

4.2.8 Clique

Recall the problem description of Clique.

Problem Description (Clique)

Given: A graph $G = (V, E)$ and a natural number k.

Question: Is there a clique of size at least k in G, i.e., is there a set C of vertices such that $G[C]$ is isomorphic to the complete graph $K_{|C|}$?

Standard Verifier:

$$V_{\text{Clique}} = \{((G, k), C) : G \text{ is a graph, } k \in \mathbb{N}, \text{ and } C \text{ is a } k\text{-clique of } G\}.$$

Since the simple standard reduction from VC to Clique is also a gp-reduction, we have the following theorem.

Theorem 4.2.15. Clique *is $\exists_r \forall_l gp$-complete via V_{Clique}.*

4.3 A list of $\exists_r\forall_l gp$-complete problems

In the last section we have shown $\exists_r\forall_l gp$-completeness via the corresponding natural verifier for the six basic NP-complete problems from [GJ79] and for a few more. Since all these typical representatives of NP-problems behave the same, we have reason to conjecture that all NP-complete problems might be $\exists_r\forall_l gp$-complete. In this section we give more arguments for this conjecture, by showing the $\exists_r\forall_l gp$-completeness of some more NP-complete problems via their natural verifiers. So, for each of the following problems we identify the "natural" verifier as universal and we show that the associated alternative solution problem is NP-complete. Again, to avoid unnecessary commentarial sentences, we only give the problem definition and the used verifier without further explanations.

4.3.1 0/1-Integer Programming (0/1-IP)

Problem Description (0/1-Integer Programming)

Given: An integer matrix $A \in \mathbb{Z}^{m\times n}$ and an integer vector $b \in \mathbb{Z}^n$.

Question: Is there a Boolean vector $x \in \{0,1\}^m$ such that $Ax = b$?

Standard Verifier:

$$V_{0/1\text{-IP}} = \{((A,b),x) : A \in \mathbb{Z}^{m\times n}, b \in \mathbb{Z}^n, x \in \{0,1\}^m, \text{ and } Ax = b\}.$$

The reduction SAT $\leq_m^p$ 0/1-Integer Programming, given in [Kar72] actually is a *gp*-reduction when we use the proper generating function g.

Theorem 4.3.1. 0/1-Integer Programming *is $\exists_r\forall_l gp$-complete via $V_{0/1\text{-IP}}$.*

4.3.2 Dominating Set (DS)

Problem Description (Dominating Set)

Given: A graph $G = (V,E)$ and a natural number k.

Question: Does G have a dominating set of size at most k, i.e., is there a subset S of V with $|S| \leq k$ such that each vertex from $V \setminus S$ is adjacent to a vertex from S ?

Standard Verifier:

$$V_{\text{DS}} = \{((G,k),S) : \text{ } G \text{ is a graph, } k \text{ is a natural number, and } S \text{ is a dominating set for } G \text{ of size } k \text{ or less}\}.$$

Theorem 4.3.2. Dominating Set *is $\exists_r\forall_l gp$-complete via V_{DS}.*

Proof. We will describe a *gp*-reduction $(\mathrm{X3C}, V_{\mathrm{X3C}}) \leq^p_{gp} (\mathrm{DS}, V_{\mathrm{DS}})$. Then, by the fact that X3C is $\exists_r \forall_l gp$-complete via V_{X3C} (Theorem 4.2.6), the assertion is immediate.

The reduction will be a combination of the *gp*-reduction from $(\mathrm{X3C}, V_{\mathrm{X3C}})$ to $(\mathrm{VC}, V_{\mathrm{VC}})$, given in the proof of Theorem 4.2.14 and a very simple reduction of VC to DS. It is important to note, that the mentioned reduction $(\mathrm{X3C}, V_{\mathrm{X3C}})$ to $(\mathrm{VC}, V_{\mathrm{VC}})$ generates VC-instances (G, k) that never have vertex covers of size $< k$.

Let (G, k) with $G = (V, E)$ be such an instance for VC generated by the above reduction. It follows that G has no vertex cover of size $< k$. Now, we will describe the reduction from VC to DS. For each edge $e = \{u, v\} \in E$, we add two new vertices, say $w_{e,1}$ and $w_{e,2}$, and the edges $\{u, w_{e,1}\}$,$\{w_{e,1}, v\}$,$\{u, w_{e,2}\}$, and $\{w_{e,2}, v\}$ and call the resulting graph $G' = (V', E')$ (see Figure 4.11). We map the VC-instance (G, k) to the instance (G', k) for Dominating Set. It is easy to see that each size k vertex cover V' of G is a size k dominating set of G'.

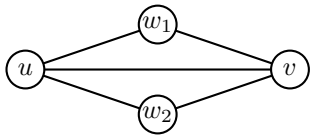

Figure 4.11: A modified edge $e = \{u, v\}$ with the additional vertices $w_{e,1}$ and $w_{e,2}$ and edges $\{u, w_{e,1}\}$,$\{w_{e,1}, v\}$,$\{u, w_{e,2}\}$, and $\{w_{e,2}, v\}$. For the illustration, we omit the lower index e from $w_{e,1}$ and $w_{e,2}$.

Conversely, let S be a size k dominating set of G'. First, assume that S includes any of the added vertices, say w.l.o.g. $w_{e,1}$, for some edge $e \in E$. If S includes any of the endpoints of e, $S \setminus \{w_{e,1}\}$ is a smaller dominating set. Otherwise, S must also contain $w_{e,2}$ and we get a smaller dominating set by deleting $w_{e,1}$ and $w_{e,2}$ from S and adding one of the endpoints of e. In both cases we obtain a dominating set S' of size $< k$ without any of the vertices $w_{e,1}$ and $w_{e,2}$, $e \in E$. It is not hard to see, that such a dominating set is also an vertex cover G of size $< k$, which is impossible. Hence, S includes none of the added vertices $w_{e,1}$ and $w_{e,2}$, $e \in E$ and we analogously have that S is a size k vertex cover of G.

So the function $f' : (G, k) \mapsto (G', k)$ maps the VC-instance generated by the reduction $(\mathrm{X3C}, V_{\mathrm{X3C}}) \leq^p_{gp} (\mathrm{VC}, V_{\mathrm{VC}})$ to instances for DS, such that $g' = id$ is a bijection between their solution sets.

Hence, combining the *gp*-reduction $(\mathrm{X3C}, V_{\mathrm{X3C}}) \leq^p_{gp} (\mathrm{VC}, V_{\mathrm{VC}})$ via f and g with the mappings f' and g', we gain a *gp*-reduction $(\mathrm{X3C}, V_{\mathrm{X3C}}) \leq^p_{gp} (\mathrm{DS}, V_{\mathrm{DS}})$ via $f \circ f'$ and $g \circ g'$. □

4.3.3 Independent Set (IS)

Problem Description (Independent Set)

Given: A graph $G = (V, E)$ and a natural number k.

Question: Is there an independent set of size at least k for G, i.e., is there a vertex set $V' \subseteq V$ such that $|V'| \geq k$ and no edge from E is incident to a vertex from V' ?

Standard Verifier:

$V_{\text{IS}} = \{((G, k), I) : G = (V, E)$ is a graph, k is a natural number, and I is an independent set of size at least k for $G\}$.

The very simple standard reduction from VC to Independent Set is also a *gp*-reduction using the standard verifiers. Thus we have the following theorem.

Theorem 4.3.3. IS *is* $\exists_r\forall_l gp$*-complete via* V_{IS}.

4.3.4 Kernel

See Section 3.2.1 for a problem description.

Theorem 4.3.4. Kernel *is* $\exists_r\forall_l gp$*-complete via* V_{Kernel}.

Proof. The many-one reduction SAT $\leq_m^p$ Kernel given in [Chv73] is actually a *gp*-reduction $(\text{SAT}, V_{\text{SAT}}) \leq_{gp}^p (\text{Kernel}, V_{\text{Kernel}})$. Since this paper seems to be unavailable, we will give the idea of the reduction found on the authors website. Then the theorem is immediate by Lemma 4.1.10 and the $\exists_r\forall_l gp$-completeness of SAT via V_{SAT} (Theorem 4.1.11).

Let F be any Boolean formula with clauses $C^1, \ldots, C^m$ and variables $x_1, \ldots, x_n$. The reducing function f will map the formula F to the graph $G = (V, A)$ that will be defined in stages in the following. First, we put the vertices x_i and $\neg x_i$, $1 \leq i \leq n$, into V and the arcs $(x_i, \neg x_i)$ and $(\neg x_i, x_i)$, $1 \leq i \leq n$, into A. Then, for each clause C^i we put three vertices c_1^i, c_2^i, c_3^i into V and the arcs (c_1^i, c_2^i), (c_2^i, c_3^i), and (c_3^i, c_1^i) into A. So we have a two-cycle $x_i \to \neg x_i \to x_i$ for each variable x_i and a three-cycle $c_1^i \to c_2^i \to c_3^i \to c_1^i$ for each clause C^i. To complete the construction of G, we furthermore add the arcs (c_1^i, u), (c_2^i, u), and (c_3^i, u) if u is a literal in the clause C^i. This concludes the construction of G.

Let α be a satisfying assignment of F. It is not hard to see that the set $S_\alpha := \{x_i : 1 \leq i \leq n \wedge \alpha(x_i) = 1\} \cup \{\neg x_i : 1 \leq i \leq n \wedge \alpha(x_i) = 0\}$ is a kernel for $f(F) = G$. So, when we define that the generating function g maps satisfying assignments α of F to $S(\alpha)$, we have that g maps satisfying assignments of F to kernels of $f(F) = G$. It is quite obvious that different assignments α and β are mapped to different sets S_α and S_β, that is, g is injective. All we furthermore have to show is that g is also surjective.

Let S be a kernel of $G = f(F)$. We must show that S is the set S_α for a satisfying assignment α of F. First, observe that S must include exactly one of

the vertices x_i and $\neg x_i$ for each i. So S induces an assignment α_S of the variables $x_1, \ldots, x_n$ via $\alpha_S(x_i) = 1$ if $x_i \in S$ and $\alpha_S(x_i) = 0$, otherwise.

Now, let C^i be a clause of F and consider the associated vertices c_1^i, c_2^i, and c_3^i. For a proof of contradiction, assume that α_S does not satisfy C^i. Then, there are no arcs from c_1^i, c_2^i or c_3^i to any of the vertices x_i or $\neg x_i$ in S. Since c_1^i, c_2^i, and c_3^i form a 3-cycle, the kernel S contains at most one of the three vertices. Otherwise, there is an arc between two kernel vertices. It follows, that there is no arc outgoing from at least one of the remaining non-kernel vertices, a contradiction. Hence, the assumption that α_S does not satisfy C^i is wrong. This argument holds for all clauses and thus, α_S satisfies all clauses from F.

So one of the literals of each clause C^i is contained in S. Consequently, there are arcs from c_1^i, c_2^i, and c_3^i to this literal. It follows that c_1^i, c_2^i, and c_3^i are not contained in the kernel S. Now, it is easy to see that $S = S_{\alpha_S}$. Hence, g is also surjective and thus, bijective.

Since f and g are polynomial-time computable, we have a gp-reduction from $(\mathrm{SAT}, V_{\mathrm{SAT}})$ to $(\textsc{Kernel}, V_{\textsc{Kernel}})$ via f and g. □

4.3.5 Knapsack (KS)

Problem Description (Knapsack)

Given: A 5-tuple (U, s, a, B, K), where U is a finite set, s (size), and v (value) are functions that assign positive natural numbers to the elements of U and B, K are natural numbers.

Question: Is there a set $U' \subseteq U$ such that $\sum_{u \in U'} s(u) \leq B$ and $\sum_{u \in U'} v(u) \geq U$?

Standard Verifier:

$$V_{\mathrm{KS}} = \{((U, s, a, B, K), U') : U \text{ is a finite set, } s, v \text{ are mappings from } U \text{ to } \mathbb{N}^+, B, K \in \mathbb{N}^+, \text{ and } U' \subseteq U \text{ with } \sum_{u \in U'} s(u) \leq B \wedge \sum_{u \in U'} v(u) \geq U\}.$$

Theorem 4.3.5. Knapsack *is* $\exists_r \forall_l gp$*-complete via* V_{KS}.

Proof. The result follows from the fact that the $\exists_r \forall_l gp$-complete problem Partition (see Theorem 4.2.13) is a restriction of Knapsack. it is not hard to see that mapping an instance (A, s) for Partition to the instance

$$(A, s, s, \frac{1}{2}\sum_{a \in A} s(a), \frac{1}{2}\sum_{a \in A} s(a)),$$

combined with the identity as generating function, realizes the gp-reduction from $(\textsc{Partition}, V_{\textsc{Part}})$ to $(\textsc{Knapsack}, V_{\mathrm{KS}})$. □

4.3.6 Minimum Edge Cost Flow (MECF)

See Section 3.2.2 for a problem description.

The proof of the NP-completeness of Minimum Edge Cost Flow [EJ77] has not been published. So we give our (possibly analogous) reduction, that is even a $\leq^p_{gp}$-reduction and thus, implies $\exists_r \forall_l gp$-completeness.

Theorem 4.3.6. MECF *is* $\exists_r \forall_l gp$*-complete via* V_{MECF}.

Proof. We give a *gp*-reduction from $(\text{X3C}, V_{\text{X3C}})$ to $(\text{MECF}, V_{\text{MECF}})$. So let $(V, \mathfrak{C})$ be an instance for X3C with $V = \{v_1, \ldots, v_n\}$ and $\mathfrak{C} = \{C_1, \ldots, C_m\}$. We assume that $|V|$ is divisible by three, otherwise, we map $(V, \mathfrak{C})$ to a fixed non-member of MECF. A correct instance is mapped to a MECF-instance (G, s, t, c, p, R, B) as described in the following. The underlaying digraph is $G = (\{s,t\} \cup \{C_1, \ldots, C_m\} \cup \{v_1, \ldots, v_n\}, \{(s, C_i) : 1 \leq i \leq m\} \cup \{(C_i, v_j) : v_j \in C_i\} \cup \{(v_j, t) : 1 \leq j \leq n\}$. The capacity function c is 3 for the arcs (s, C_i), $1 \leq i \leq m$, and one for the remaining arcs. The price p for the arcs (s, C_i), $1 \leq i \leq m$, is one and zero for all other arcs[4]. The requirement R is n and the price bound B is $n/3$ (see Figure 4.12 for an example).

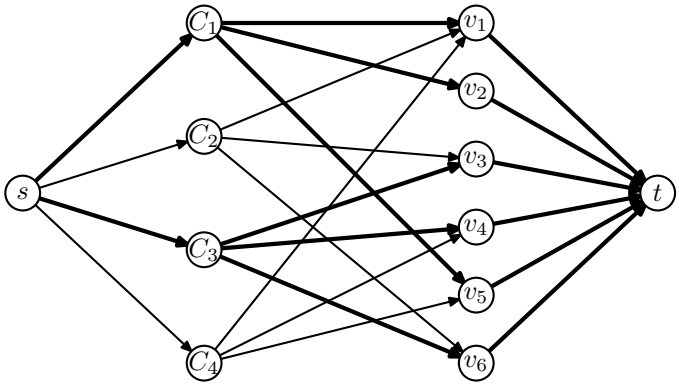

Figure 4.12: The flow problem for the X3C-instance $(\{v_1, \ldots, v_6\}, \{\{v_1, v_2, v_5\}, \{v_1, v_3, v_6\}, \{v_3, v_4, v_6\}, \{v_1, v_4, v_5\}\}$. The requirement R is 6 and the price bound B is 2. The bold arcs represent a valid flow function, when all marked arcs are uses with maximal capacity. The associated X3C-solution is $\{\{v_1, v_2, v_5\}, \{v_3, v_4, v_6\}\}$.

It is easy to see that due to the requirement $R = n$ all arcs (v_j, t), $1 \leq j \leq n\}$ and exactly n of the arcs $\{(C_i, v_j) : v_j \in C_i\}$ must be used with flow 1. Due to the price bound at most $n/3$ of the n arcs $\{(s, C_i) : 1 \leq i \leq m\}$ can be used. Since

[4]Note that formally the prize zero is forbidden. Since for each solution the overall number of used network arcs is $n/3 + n + n$, we could add one to all prices and $7/3 \cdot n$ to B to smooth out this problem.

their capacity is three, exactly $n/3$ of them must be used and the flow along them must be three. We denote the subcollection of $\mathfrak{C}$, induced by such $n/3$ chosen arcs, by $\mathfrak{C}'$.

Following this, it is not hard to see that a valid flow function induces a subcollection $\mathfrak{C}'$, which is an exact 3-cover of V and an exact 3-cover also induces a valid flow function. Since this relation is bijective, we have $(\text{X3C}, V_{\text{X3C}}) \leq^p_{gp} (\text{MECF}, V_{\text{MECF}})$. □

4.3.7 Set Packing (SP)

Problem Description (Set Packing)

Given: A family $\mathcal{S} = \{S_1, \ldots, S_m\}$ of sets and a natural number ℓ.

Question: Are there ℓ pairwise disjoint sets in $\mathcal{S}$?

Standard Verifier:

$$V_{\text{SP}} = \{((\mathcal{S}, \ell), I) : \mathcal{S} = \{S_1, \ldots, S_m\} \text{ is a family of sets}, \ell \in \mathbb{N}, \text{ and } I \text{ is a set of } \ell \text{ indices } i_1, \ldots, i_\ell \text{ such that } S_{i_j} \cap S_{i_k} = \varnothing, 1 \leq j \neq k \leq \ell\}.$$

Theorem 4.3.7. Set Packing *is $\exists_r \forall_l gp$-complete via V_{SP}.*

Proof. The proof is a simple reduction from X3C. It is easy to see that the mapping $(V, \mathfrak{C}) \mapsto (\mathfrak{C}, V/3)$ combined with the proper generating function realizes the reduction $(\text{X3C}, V_{\text{X3C}}) \leq^p_{gp} (\text{SP}, V_{\text{SP}})$. □

4.3.8 Shortest Weight Constrained Path (SWCP)

Problem Description (Shortest Weight Constrained Path)

Given: A 7-tuple (G, s, t, ℓ, w, L, W), where $G = (V, E)$ is a graph, s (source) and t (sink) are specified vertices, ℓ (length) and w (weight) are mappings from the edge set E to positive integers.

Question: Is there a simple path from s to t with total weight W or less and total length L or less ?

Standard Verifier:

$$V_{\text{SWCP}} = \{((G, s, t, \ell, w, L, W), P) : \ G = (V, E) \text{ is a graph}, s, t \in V, \ \ell \text{ and } w \text{ are functions from } E \text{ to } \mathbb{N}^+, \ L, W \in \mathbb{N}^+, \text{ and } P \text{ is a simple path from } s \text{ to } t \text{ with total weight } W \text{ or less and total length } L \text{ or less}\} .$$

To show the NP-completeness of SHORTEST WEIGHT CONSTRAINED PATH, Wang and Crowcroft gave a many one-reduction from PARTITION [WC96]. Since this reduction is also generating parsimonious, we have the following theorem by the $\exists_r \forall_l gp$-completes of PARTITION via V_{PART} .

Theorem 4.3.8. SHORTEST WEIGHT CONSTRAINED PATH *is $\exists_r \forall_l gp$-complete via V_{SWCP}.*

4.3.9 Steiner Tree (ST)

Problem Description (Steiner Tree)

Given: A graph $G = (V, E)$, a set $R \subseteq V$ of nodes, a weight function $w : E \to \mathbb{N}$, and a bound $B \in \mathbb{N}$.

Question: Is there a subtree T of G that includes all vertices from R such that the edge-weight-sum of T is B or less ?

Standard Verifier:

$$V_{\text{ST}} = \{((G, R, w, B), T) : G = (V, E) \text{ is a graph, } R \subseteq V, w \text{ maps } E \text{ to } \mathbb{N}, \\ B \in \mathbb{N}, \text{ and } T \text{ is a subtree with the above properties}\}.$$

In [Kar72] STEINER TREE is shown to be NP-complete by a reduction from EXACT COVER (see [GJ79] for a definition), which can easily be modified to also be generating parsimonious. Since EXACT COVER is a generalization of X3C, it follows that X3C $\leq^p_{gp}$ STEINER TREE, which implies the following theorem.

Theorem 4.3.9. STEINER TREE *is $\exists_r \forall_l gp$-complete via V_{ST}.*

Note that this result implies the NP-completeness of all alternative solution problems of STEINER TREE, which negatively answers our introductory question from Chapter 1. However, we also refer to Chapter 7, where we give a positive result for STEINER TREE in terms of approximability.

4.3.10 Traveling Salesman Problem (TSP)

Problem Description (Traveling Salesman Problem)

Given: A graph $G = (V, E)$, a weigh function $w : E \to \mathbb{N}$, and a bound $B \in \mathbb{N}$.

Question: Is there a Hamiltonian cycle C in G such that $\sum_{e \in C} w(e) \leq B$?

Standard Verifier:

$$V_{\text{TSP}} = \{((G, w, B), C) : G = (V, E) \text{ is a graph, }, w \text{ maps } E \text{ to } \mathbb{N}, B \in \mathbb{N}, \\ \text{and } C \text{ is a Hamiltonian cycle of } G \text{ such that } \sum_{e \in C} w(e) \leq B\}.$$

Since the $\exists_r \forall_l gp$-complete problem HAMILTONIAN CYCLE is a restriction of TSP, HC can easily be reduced to TSP.

Theorem 4.3.10. *The* TRAVELING SALESMAN PROBLEM *is $\exists_r \forall_l gp$-complete via V_{TSP}.*

Proof. It is easy to see that the functions $f : G \mapsto (G, one, |V|)$, where $one(e) = 1$, for all $e \in E$, and $g = id$ realize the sufficient reduction $(\mathrm{HC}, V_{\mathrm{HC}}) \leq^p_{gp} (\mathrm{TSP}, V_{\mathrm{TSP}})$. □

4.4 An extraordinary problem - Hamiltonian cycles in cubic graphs

All NP-complete problems treated so far, behave the same concerning the complexity of alternative solutions. For each treated problem A with its natural verifier V_A we managed to show that all problems of alternative solutions $n + A$ are NP-complete. Furthermore, we showed that the used natural verifiers are also universal. The problem of finding a Hamiltonian cycle in a cubic graph (CUBIC HAMILTONIAN CYCLE) seems to have an exceptional position, which will be explained and discussed in this section.

Problem Description (Cubic Hamiltonian Cycle)

Given: A cubic graph $G = (V, E)$, i.e., a graph such that $d_G(v) = 3$, for all $v \in V$.

Question: Is there a Hamiltonian cycle in G ?

Standard Verifier:

$$V_{\mathrm{CHC}} = \{(G, C) : G \text{ is a cubic graph and } C \text{ is a Hamiltonian cycle of } G.\}.$$

It was shown that this restriction of HAMILTONIAN CYCLE to cubic graphs is also NP-complete [GJT76]. Note furthermore that the verifier V_{CHC} is almost the same as the natural and universal verifier V_{HC} for HAMILTONIAN CYCLE. The only difference is the added requirement that the graph has to be cubic. Since the verifier V_{CHC} looks quite natural we would expect that CHC with V_{CHC} behaves like all the other examined NP-complete problems. But it does not. Tutte has shown that CHC has an interesting property.

Theorem 4.4.1 (Tutte's Theorem [Tut46]). *Each Hamiltonian cubic graph contains at least three Hamiltonian cycles.*

It follows that both problems, $1 + \mathrm{CHC}$ and $2 + \mathrm{CHC}$ are more or less trivial because each cubic graph with one or two Hamiltonian cycles contains another one. So $1 + \mathrm{CHC}$ and $2 + \mathrm{CHC}$ are in P and even in much smaller complexity

classes. So assuming that P $\neq$ NP, which is widely believed, the problem CHC contradicts the conjecture, that the natural verifiers of all NP-complete problems are also universal verifiers and that the corresponding ASP's are NP-complete.

In the remainder of this section we will discuss, if these alternative Hamiltonian cycles can be computed relatively easy, e.g. by an FP-function. Furthermore, we will see that V_{HC} is not a universal verifier and we will try to find a universal verifier.

4.4.1 Computing a second Hamiltonian cycle

The starting question of this thesis was, if a given solution of an NP-complete problem helps to find alternative solutions of this problem.

It turned out that even the seemingly easier task, to decide if there exist alternative solutions, is NP-complete for many NP-complete problems. Now we found a problem, for which this task is easy. Alternative solutions always exist for for $1 + \mathrm{CHC}$ and $2 + \mathrm{CHC}$. So it makes sense to ask for a fast algorithm that, given a cubic graph G and one (two) Hamiltonian cycle(s) of G, computes a second (a third) Hamiltonian cycle of G.

The proof of Tutte's Theorem is nonconstructive and provides no algorithm that computes a second or third Hamiltonian cycle. Such an algorithm was given by Thomason in 1978 [Tho78]. We will give the idea of this algorithm. Starting point is the given Hamiltonian cycle $C = (v_1, v_2, \ldots, v_n, v_1)$ of a cubic graph G. Since G is cubic, there exists an edge $\{v_n, v_i\}, i \neq 1, i \neq n$ incident with v_n that is not contained in C (see Figure 4.13).

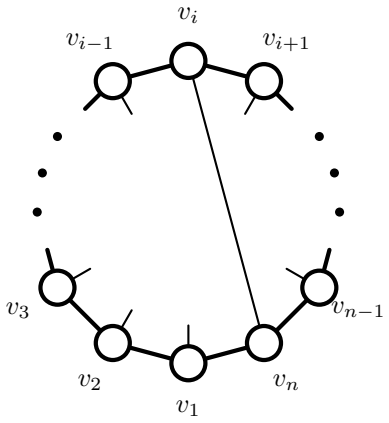

Figure 4.13: One step of Thomason's Algorithm on a graph with n vertices: remove $\{v_i, v_{i+1}\}$ and add $\{v_i, v_n\}$ to obtain the new Hamiltonian path $(v_1, v_2, \ldots, v_i, v_n, v_{n-1}, \ldots, v_{i+1})$ from $(v_1, \ldots, v_n)$.

Hence, G contains a Hamiltonian path $H = (v_1, v_2, \ldots, v_i, v_n, v_{n-1}, \ldots, v_{i+1})$. Besides the edges $\{v_i, v_{i+1}\} \in H$ and the edge $\{v_{i+1}, v_{i+2}\}$ that was just deleted from H there is a third edge $\{v_{i+1}, v_j\}$ outgoing from v_{i+1} in G. If $j = 1$ we get a

second Hamiltonian cycle by adding the edge $\{v_i, v_{i+1}\}$ to H. Otherwise, we get another Hamiltonian path of G by adding the edge $\{v_{i+1}, v_j\}$ to H and deleting one of the edges adjacent to v_j. In this manner, the algorithm steps from one Hamiltonian path to another until a Hamiltonian cycle is found. We omit the proof of the correctness of the algorithm.

Unfortunately, Thomason's algorithm does not work in polynomial time. In [Kra99] it was shown by giving a family of counterexamples that the algorithm has an exponential worst-case runtime. Even though many researches have tried to find a polynomial-time algorithm for this interesting problem, to date, there is no such algorithm known and the question for the existence of an algorithm that computes an alternative Hamiltonian cycle in a cubic graph, is still open.

4.4.2 Alternative verifiers for CHC

When considering CHC it seemed to be a natural choice to use the verifier V_{CHC} that arises from the natural (and universal) verifier for HC by restricting it to cubic graphs. With respect to this verifier we observed the property that $1+\text{CHC}$ and $2+\text{CHC}$ are easy. Maybe this property is not typical for the problem CHC but for the chosen verifier V_{CHC}. To approach this issue we will have a look at a restricted version r-SAT of SAT, which is similarly to CHC, a restriction of an NP-complete problem. Equipped with a restricted version $V_{\text{r-SAT}}$ of the universal verifier V_{SAT} for SAT, r-SAT also seems to behave like CHC. In case of r-SAT we will show that $V_{\text{r-SAT}}$ is not universal and we will give an alternative verifier that is universal and that induces NP-complete ASP's. Finding a universal verifier for r-SAT will reveal that r-SAT also behaves like all other treated NP-complete problems besides CHC. We will try to transfer the argumentation for r-SAT to CHC, that is, we will check whether V_{CHC} is universal for CHC or not and if not, we will try to find a universal verifier for CHC.

A restricted version of SAT that almost behaves like CHC

The problem CHC is a restriction of the standard NP-complete problem Hamiltonian Cycle. In the following we try to understand the special properties of CHC by discussing a restriction of another standard NP-complete problem, namely SAT, that behaves similar.

The idea of the upcoming definition for restricted-SAT (r-SAT) is to make sure that each satisfiable formula (of the restricted type) has at least two satisfying assignments. We achieve this, by forcing each formula F to have a variable, called y, whose assignment does not affect the truth-value of F. Recall that we describe clauses of CNF-formulas as sets of literals.

Problem Description (r-SAT)

Given: A CNF-formula F of the form $F = \{C \cup \{y\} : C \in \mathfrak{C}\} \cup \{C \cup \{\neg y\} :$

$C \in \mathfrak{C}\}$, where $\mathfrak{C}$ is a collection of clauses and y is a labelled variable[5] that appears in no clause of $\mathfrak{C}$.

Question: Is F satisfiable ?

Standard Verifier:

$$V_{\text{r-SAT}} = \{(F, \alpha) : F \text{ is a r-SAT-formula and } \alpha \text{ satisfies } F\}.$$

We call a formula of the above structure an r-SAT-*formula*.

The reason for demanding the variable y to be labelled is, that given such a formula, an algorithm for V_{rsat} must be able to uniquely recognize which variable plays the role of this variable y in order to check if the given formula is an r-SAT-formula.

It is easy to see that r-SAT is NP-complete, which means that the restriction does not simplify the problem.

Theorem 4.4.2. r-SAT *is* NP*-complete.*

Proof. We give a $\leq_m^p$-reduction from SAT. Let F be a formula in CNF having the clauses $\mathfrak{C}$. It is easy to see, that the formula $f(F)$ consisting of the clauses $\{C \cup \{y\} : C \in \mathfrak{C}\} \cup \{C \cup \{\neg y\} : C \in \mathfrak{C}\}$, where y is labelled, is an r-SAT-formula, which is satisfiable if and only if F is satisfiable. Thus, SAT $\leq_m^p$ r-SAT via f. □

Similarly to CHC, it is trivial to decide whether an r-SAT-formula with a given satisfying assignment, has an alternative satisfying assignment.

Theorem 4.4.3. $1 +$ r-SAT *(via* $V_{\text{r-SAT}}$*) is in* P*.*

Proof. Let (F, α) be an instance for $1 +$ r-SAT with the labelled variable y. If F is no r-SAT-formula or $F(\alpha) = false$, (F, α) can be rejected. Otherwise, (F, α) is accepted, since we gain an alternative solution from α by flipping the value of y. □

Note that, $|V_{\text{r-SAT}}(F)|$ is always even. So, r-SAT also has easy ASP's. We can solve this contradiction to our conjecture of hard ASP's by applying the notion of universal verifiers. It turns out, that $V_{\text{r-SAT}}$ is not universal.

Theorem 4.4.4. $V_{\text{r-SAT}}$ *is not universal for* r-SAT*.*

Proof. Consider the following verifier $V'_{\text{r-SAT}}$ for r-SAT,

$$\begin{aligned} V'_{\text{r-SAT}} = \{&(F, \alpha) : F \text{ is a satisfying r-SAT-formula over the variable set } X \\ &\text{and the special variable } y \text{ and } \alpha \text{ is the restriction of a satisfying} \\ &\text{assignment of } F \text{ to } X \setminus \{y\}\}. \end{aligned}$$

[5]For instance, the way of labelling the variable y could be that y is the variable that appears first in the encoding of F.

It is easy to see that $V'_{\text{r-SAT}}$ actually is a verifier for r−SAT. Now, consider the formula $F = (x \vee y) \wedge (x \vee \neg y)$, where y is labelled. The r−SAT-formula F has exactly two satisfying assignments, namely $(x, y) \mapsto (1, 0)$ and $(x, y) \mapsto (1, 1)$. With respect to $V'_{\text{r-SAT}}$ the only solution for F is $x \mapsto 1$. So, $|V'_{\text{r-SAT}}(F)| = 1$. Since $|V_{\text{r-SAT}}(F')|$ is always even, there is no function that maps F to a formula F' with $|V_{\text{r-SAT}}(F')| = 1$. Hence, $(\text{r−SAT}, V'_{\text{r-SAT}})$ is not *gp*-reducible to $(\text{r−SAT}, V_{\text{r-SAT}})$ and thus, $V_{\text{r-SAT}}$ is not universal for r−SAT. □

The above defined verifier $V'_{\text{r-SAT}}$ is a new candidate for being a universal verifier for r−SAT. The next theorem shows that $V'_{\text{r-SAT}}$, in fact, is universal.

Theorem 4.4.5. *$V'_{\text{r-SAT}}$ is universal for* r−SAT.

Proof. It suffices to show that $(\text{SAT}, V_{\text{SAT}})$ is *gp*-reducible to $(\text{r−SAT}, V'_{\text{r-SAT}})$. Then, the claim follows from Theorem 4.1.11 and Lemma 4.1.9.

A closer look at the $\leq^p_m$-reduction, given in the proof of Theorem 4.4.2 shows that this reduction is actually a *gp*-reduction, if we use the identity as generating function. □

Now, by Corollary 4.1.13, we have the NP-completeness of $n + \text{r−SAT}$ (defined w.r.t. $V'_{\text{r-SAT}}$), for all natural numbers n. So, the ASP's defined with respect to the universal verifier are NP-hard.

Corollary 4.4.6. *For each natural number n, $n + \text{r−SAT}$ (defined w.r.t. $V'_{\text{r-SAT}}$) is* NP*-complete.*

Is there a universal verifier for CHC?

Now, we try to translate the argumentation for r−SAT to CHC. In a first step we show that V_{CHC} is not a universal verifier for CHC.

Theorem 4.4.7. *The verifier V_{CHC} is not a universal verifier for* CHC.

Proof. Consider the verifier V'_{CHC} for CHC which is defined as follows:

$$V'_{\text{CHC}} = \{(G, \mathfrak{C}) : G \text{ is a cubic graph and } \mathfrak{C} = \{C_1, C_2, C_3\} \text{ such that } C_1, C_2, \text{ and } C_3 \text{ are pairwise different Hamiltonian cycles of } G\}.$$

By Tutte's Theorem 4.4.1 it follows that V'_{CHC} is a verifier for CHC.

For a proof by contradiction assume that V_{CHC} is a universal verifier for CHC. Thus, $(\text{CHC}, V'_{\text{CHC}})$ is *gp*-reducible to $(\text{CHC}, V_{\text{CHC}})$ via functions f and g. Consider the K_4, the complete graph with four vertices which is obviously a cubic graph. Note that the K_4 has exactly three Hamiltonian cycles, say C_1, C_2, and C_3. It follows that $V'_{\text{CHC}}(K_4) = \{\{C_1, C_2, C_3\}\}$, i.e., there is exactly one V'_{CHC}-solution for the K_4. Since g is a bijective mapping between $V'_{\text{CHC}}(K_4)$ and $V_{\text{CHC}}(f(K_4))$, it follows that there exists a cubic graph $f(K_4)$ having exactly one Hamiltonian cycle, a contradiction to Theorem 4.4.1. Thus the assumption is wrong and V_{CHC} is not universal. □

The above proof provides a new candidate for a universal verifier for CHC, namely V'_{CHC}. In V'_{CHC} three Hamiltonian cycles are merged to one solution, which reflects that each Hamiltonian cubic graph has three Hamiltonian cycles. However, a counting argument applied to a cubic graph with more than three Hamiltonian cycles shows that V'_{CHC} is not universal.

Theorem 4.4.8. *The verifier V'_{CHC} is not a universal verifier for* CHC*.*

Proof. We show that $(\mathrm{CHC}, V_{\mathrm{CHC}})$ is not $\leq^p_{gp}$-reducible to $(\mathrm{CHC}, V'_{\mathrm{CHC}})$. Observe that a cubic graph G with $k \geq 3$ Hamiltonian cycles has exactly $\binom{k}{3}$ V'_{CHC}-solutions, namely all three-element sets of Hamiltonian cycles of G. Hence, there is no cubic graph with exactly six V'_{CHC}-solutions, because $\binom{k}{3} \neq 6$, for all $k \in \mathbb{N}$. Analogous to the proof of Theorem 4.4.7 it suffices to give a cubic graph G with exactly six V_{CHC}-solutions in order to prove that $(\mathrm{CHC}, V_{\mathrm{CHC}})$ is not *gp*-reducible to $(\mathrm{CHC}, V'_{\mathrm{CHC}})$ because this graph can not be mapped to a graph having six V'_{CHC}-solutions. Consider the following graph, which is well-know as $K_{3,3}$.

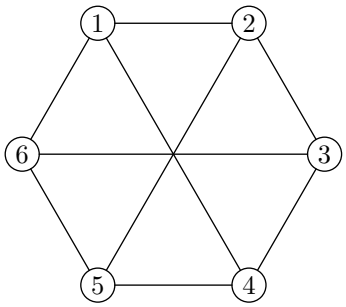

Figure 4.14: The $K_{3,3}$ - a cubic graph G with exactly six Hamiltonian cycles.

It is a simple task to verify that the graph $K_{3,3}$ (see Figure 4.14) is cubic and has exactly six Hamiltonian cycles, namely

1. $C_1 = \{\{1,2\},\{2,3\},\{3,4\},\{4,5\},\{5,6\},\{6,1\}\}$,
2. $C_2 = \{\{1,2\},\{2,3\},\{3,6\},\{6,5\},\{5,4\},\{4,1\}\}$,
3. $C_3 = \{\{1,2\},\{2,5\},\{5,4\},\{4,3\},\{3,6\},\{6,1\}\}$,
4. $C_4 = \{\{1,2\},\{2,5\},\{5,6\},\{6,3\},\{3,4\},\{4,1\}\}$,
5. $C_5 = \{\{1,4\},\{4,3\},\{3,2\},\{2,5\},\{5,6\},\{6,1\}\}$, and
6. $C_6 = \{\{1,4\},\{4,5\},\{5,2\},\{2,3\},\{3,6\},\{6,1\}\}$.

Thus, $(\mathrm{CHC}, V_{\mathrm{CHC}})$ is not *gp*-reducible to $(\mathrm{CHC}, V'_{\mathrm{CHC}})$. □

So we have shown that neither V_{CHC} nor V'_{CHC} are universal verifiers for CHC. Both proofs were established using the fact that for both verifiers exist natural numbers such that no instance has the respective number of solutions. It is reasonable to assume that a potential universal verifier for CHC should not be vulnerable to such counting arguments. So, a verifier that directly refers to the notion of Hamiltonian cycles is probably not a good candidate.

To achieve an alternative characterization assume that a cubic graph $G = (V, E)$ has a Hamiltonian cycle C. Color all edges not in C with color 1 and alternatively color the edges from C with 2 and 3. It is not hard to see, that this coloring is a valid 3-coloring of the edges of G. Conversely, we call a valid 3-coloring c of G *Hamiltonian* if one of the sets $C_1 \cup C_2$, $C_1 \cup C_3$, or $C_2 \cup C_3$ is a Hamiltonian cycle of G, where $C_i = \{e \in E : c(e) = i\}$, $i \in \{1, 2, 3\}$. It is not hard to see, that the verifier

$$V''_{\text{CHC}} = \{G, \{C_1, C_2, C_3\} : G \text{ is a cubic graph and } \{C_1, C_2, C_3\} \text{ describes a Hamiltonian 3-coloring of } G\}$$

actually is a verifier for CHC. By the following Lemma we have that universality of V''_{CHC} can not be ruled out using a simple counting argument.

Lemma 4.4.9. *For each natural number n, there exists a cubic graph G_n such that $|V''_{\text{CHC}}(G_n)| = n$.*

Proof. It can easily be verified, that the graphs G_0 and G_1 from Figure 4.15 satisfy $|V''_{\text{CHC}}(G_n)| = n$ for $n = 0$ and $n = 1$.

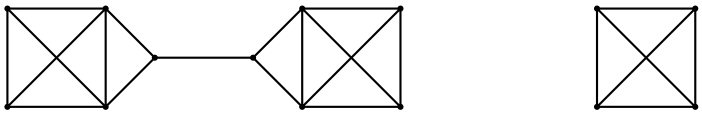

Figure 4.15: The graphs G_n for $n = 0$ (left) and $n = 1$ (right).

Now, consider the family H_i, $i \geq 3$, of graphs as depicted in Figure 4.16.

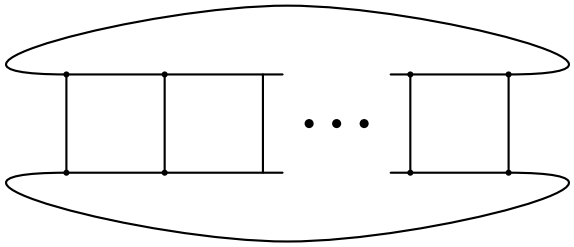

Figure 4.16: The family of graphs H_i, $i \geq 3$, where i denotes the number of concatenated 4-cycles.

It is not hard to verify, that all graphs H_{2j}, $j \geq 2$ have exactly $(2j+1)$ Hamiltonian colorings, namely the ones induced by the Hamiltonian cycles depicted in Figure 4.17.

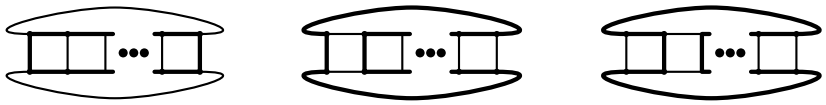

Figure 4.17: Hamiltonian colorings of H_{2j}. Analogous to the two rightmost cycles, there are $2j-2$ more cycles.

Furthermore, it is not hard to see that H_3 has exactly three Hamiltonian colorings. Thus, we have a graph with $|V''_{\text{CHC}}(G_n)| = n$, for $n = 0$ and each odd natural number. We gain such a graph for all remaining even natural numbers, by replacing one edge by the subgraph from Figure 4.18, that doubles the number of Hamiltonian colorings.

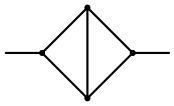

Figure 4.18: A subgraph such that, if any edge is replaced by this subgraph, the number of Hamiltonian colorings is doubled.

Thus, we have a graph G_n with $|V''_{\text{CHC}}(G_n)| = n$, for each natural number n. □

So, V''_{CHC} is a very good candidate for being a universal verifier for CHC. Unfortunately, despite spending enormous efforts in trying to find a *gp*-reduction $(A, V_A) \leq^p_{gp} (\text{CHC}, V''_{\text{CHC}})$ for a $\exists_r\forall_l gp$-complete problem A with a universal verifier V_A, this problems remains open. Also a weaker result, like the NP-completeness for $n + \text{CHC}$ (defined via V''_{CHC}), for some natural number $n > 0$, is an interesting open question. Nevertheless, we believe in the latter and in the universality of V''_{CHC}.

4.5 Conclusions

In the above sections we have shown the $\exists_r\forall_l gp$-completeness for a long list of NP-complete problems via their natural verifiers. By Corollary 4.1.13 we have the following corollary.

Corollary 4.5.1. *The natural verifiers of the problems* 0/1-Integer Programming, 3Dimensional Matching, 3Satisfiability, Clique, Dominating Set, Exact-3-Cover, Hamiltonian Cycle, Independent Set, Kernel, Knapsack, Minimum Edge Cost Flow, Partition, SAT, Set Packing,

Shortest Weight Constrained Path, Steiner Tree, Tiling, Traveling Salesman Problem, *and* Vertex Cover *are universal and the corresponding alternative solution problems are* NP*-complete.*

For all of these problems we exhaustively answered the question for the complexity of alternative solutions. We have shown that all the natural verifiers are also universal and that the associated ASP's are NP-complete. Thus, we might conjecture that this holds for all NP-complete problems. Since the natural verifier is an informal and subjective notion we abstain from formulating this conjecture. However, we state the following conjecture, that does not include the notion of natural verifiers.

Conjecture 4.5.2. *For each* NP*-complete problem A, there exists a universal verifier V_A such that A is $\exists_r \forall_l gp$-complete via V_A and thus, $n + A$ (defined w.r.t. V_A) is* NP*-complete, for all natural numbers n.*

Even though we believe in this conjecture, there is at least one problem that resists to fit into this scheme, namely the problem of finding Hamiltonian cycles in cubic graphs. Tutte proved in [Tut46] that each Hamiltonian cubic graph has at least three Hamiltonian cycles. So, when treating Hamiltonian cycles as solutions, $1 + \mathrm{CHC}$ and $2 + \mathrm{CHC}$ are obviously in P. However, the question for a polynomial-time algorithm that finds those alternative Hamiltonian cycle is still open.

Open Problem 1. *Is there a polynomial-time algorithm computing alternative Hamiltonian cycles for cubic graphs?*

By counting arguments, we have seen that the natural verifier V_{CHC} for CHC, that treats Hamiltonian cycles as solutions, is not a universal verifier. However, we gave a promising candidate V''_{CHC} for a universal verifier, by which Hamiltonian 3-colorings are solutions. We have shown that V''_{CHC} is not vulnerable to such counting arguments but the questions if V''_{CHC} is universal and if all (or some) of the problems $n + \mathrm{CHC}$ (defined via V''_{CHC}), $n \in \mathbb{N}$, are NP-complete remain open.

Open Problem 2. *Find out if or if not V''_{CHC} is universal for* CHC*! In case of not, look for another candidate!*

Maybe the answer to Open Problem 2 also answers the following one.

Open Problem 3. *Verify or disprove Conjecture 4.5.2!*

Chapter 5

Alternative Solutions in RE

The main part of this thesis deals with the question of alternative solutions of NP-problems. It suggests itself to also ask this question for other complexity classes than NP. The basic requirement to such a class is the existence of a notion of solutions for its problems. In the case of NP this notion is provided by Theorem 3.1.1 and since we have a very similar theorem for RE (see Theorem 5.1.1 below), it is reasonable to have a closer look at alternative solutions of RE-problems.

5.1 Preliminaries

In this section we slightly alter our definition of Turing machines. Here, all Turing machines have exactly one halting state, namely an accepting one. We say that a Turing machine M rejects an input x if and only if the computation $M(x)$ never halts.

When dealing with RE it is common to make use of a polynomial-time computable and polynomial-time invertible bijection between Σ^* and $\mathbb{N}$ for finite alphabets Σ. So in this section, we make no difference between natural numbers and strings from Σ^*, as inputs for Turing machines. We also assume that the states of Turing machines are denoted by natural numbers.

We start with the already mentioned characterization of RE, which provides a notion of solutions for RE-problems.

Theorem 5.1.1 ([Rog67]). *A problem $A \subseteq \Sigma^*$ is in* RE *if and only if there exists a problem $B \in$ REC such that, for all $x \in \Sigma^*$,*

$$x \in A \leftrightarrow (\exists\, y \in \Sigma^*)[(x, y) \in B)].$$

Adopting the NP-concepts of verifiers and solutions, the REC-problem B plays the role of a verifier and a word $y \in \Sigma^*$ with $(x, y) \in B$ is a solution for x with respect to B. Since the notions of verifiers, *gp*-reduction, $\exists_r\forall_l gp$-reduction, $\exists_r\forall_l gp$-completeness and universal verifier were very helpful in discussing alternative solutions for NP-problems, we would expect similar notions to be helpful for RE-problems. In fact, these notions and the surrounding theory can easily be transferred to RE. The starting point is the notion of a verifier for RE.

Definition 5.1.2. *1. A relation $V \subseteq \Sigma^* \times \Sigma^*$ is called an* RE-verifier *if and only if $V \in$ REC.*

2. Let V be an RE*-verifier. We define the problem $L(V)$ that is associated with V, via*

$$L(V) = \{x \in \Sigma^* : (\exists\ y \in \Sigma^*)[(x, y) \in V]\}.$$

3. For an RE*-verifier V and a word $x \in \Sigma^*$, $V(x)$ denotes the* set of solutions for x with respect to V, *that is,*

$$V(x) = \{y \in \Sigma^* : (x, y) \in V\}.$$

Note that by Theorem 5.1.1 it follows that each RE-problems has an RE-verifier.

After formalizing the notion of a solution for an RE-problem A (with respect to an RE-verifier V_A) we can state the problem of alternative solutions.

Definition 5.1.3. *Let A be a problem in* RE *with an* RE*-verifier V_A and let n be a natural number. The* alternative solution problem *$n + A_{V_A}$ for A is defined as follows:*

$$\begin{aligned} n + A_{V_A} = \{&(x, y_1, \ldots, y_n) : (\forall\ 1 \leq i \leq n)[V_A(x, y_i)], (\forall\ 1 \leq i \neq j \leq n)[y_i \neq y_j], \\ &\text{and } (\exists\ y_{n+1})[(\forall\ 1 \leq i \leq n)[y_i \neq y_{n+1}] \wedge V_A(x, y_{n+1})]\}. \end{aligned}$$

When it is clear which verifier V_A is used for a problem A (which will normally be the case), we write $n + A$ instead of the more complicated notion $n + A_{V_A}$. The properties of "+"stated in Theorem 3.1.5, Theorem 3.1.6 and Corollary 3.1.7 can easily be transferred from NP to RE. We abstain from repeating these assertions.

Analogous to the very useful concept of *gp*-reduction for NP-problems (see Definition 3.3.1), we can define a notion of *gp*-reduction for RE-problems.

Definition 5.1.4. *Let A and B be problems from* RE *with associated verifiers V_A and V_B. We say that (A, V_A) is* generating parsimoniously reducible *to (B, V_B), $(A, V_A) \leq_{gp} (B, V_B)$, if and only if there exists a pair (f, g) of recursive functions $f : \Sigma^* \to \Sigma^*$ and $g : \Sigma^* \times \Sigma^* \to \Sigma^*$ such that:*

1. $x \in A \leftrightarrow f(x) \in B$ holds, for each $x \in \Sigma^*$, and

2. g_x, defined via $g_x(y) = g(x, y)$, is a bijection between $V_A(x)$ and $V_B(f(x))$, for each $x \in \Sigma^*$.

Because of the strong analogy to the case of NP, it is easy to see that we gain the same properties as in the case of NP.

Theorem 5.1.5. *Let A be a problem in* RE *and let V_A be a verifier for A. If A is gp-reducible to $1 + A$, then A is gp-reducible to $n + A$, for each natural number n.*

Proof. The proof is basically the same as for Theorem 3.3.5. □

Theorem 5.1.6. *Let A and B be problems in* RE *with associated* RE*-verifiers V_A and V_B. If $A \leq_{gp} B$, then $n + A \leq_{gp} n + B$, for each natural number n.*

Proof. The proof can easily be transferred from Theorem 3.3.9. □

Note that $n + A \leq_{gp} n + B$ implies $n + A \leq_m n + B$.

Similar to the case of NP, it is reasonable to ask which is the "right" choice of an RE-verifier for an RE-problem. For NP, we gave a possible answer using the concept of $\exists_r \forall_l gp$-reduction, universal verifiers and $\exists_r \forall_l gp$-completeness. We just mention that these notions can also easily be transferred from NP to RE and abstain from repeating their definitions.

5.2 Alternative solutions and universal verifiers in RE

In NP we had the following situation. We managed to show that all problems $n + \mathrm{SAT}$, $n \in \mathbb{N}^+$, (defined with respect to some standard verifier V_{SAT}) are NP-complete. Furthermore, we proved $\exists_r \forall_l gp$-completeness of SAT for NP. Thus, proving that $(\mathrm{SAT}, V_{\mathrm{SAT}})$ is gp-reducible to (A, V_A) is sufficient to gain $\exists_r \forall_l gp$-completeness of the problem A via V_A, the universality of V_A for A and the NP-completeness of all $n + A$, defined via the universal verifier V_A. Since, all these notions and corresponding properties carry over to RE, we are in the same comfortable situation, if we manage to find a starting point for RE like SAT in the case of NP.

One of the most common RE-problems and a first candidate is the halting problem K:

Problem Description (halting problem K)

Given: A pair (i, x) of natural numbers.

Question: Does the computation $M_i(x)$, i.e., the computation of the machine with the Gödel number i on the input x, ever halt ?

Standard Verifier:

$$V_K = \{((i, x), k) : \text{the computation } M_i(x) \text{ halts after } k \text{ computational steps}\}.$$

It is well-known that the halting problem K is RE-complete.

It turns out that the halting problem K does not behave very conveniently, so we first consider the following problem K' instead, which is a (slightly) generalized version of the halting problem K and we return to the standard halting problem K later.

Problem Description (generalized halting problem K')

Given: A 3-tuple (i, x, j) of natural numbers.

Question: Does the computation $M_i(x)$ ever reach the state j ?

Standard Verifier:

$V_{K'} = \{((i, x, j), k) :$ after k computational steps the computation of $M_i(x)$ is in state $j\}$.

If j denotes the accepting halting state of the machine M_i, then we have that $(i, x, j) \in K'$ if and only if M_i halts on the input x - the standard halting problem. So, we have a many-one reduction $K \leq_m K'$, via $f(i, x) := (i, x, j)$, where j denotes the accepting halting state of M_i. Hence, it is immediate that K' is also RE-complete (with respect to many-one reductions). It turns out, that K' is also $\exists_r \forall_l gp$-complete for RE via $V_{K'}$.

Theorem 5.2.1. *The generalized halting problem K' is $\exists_r \forall_l gp$-complete via $V_{K'}$ for* RE*.*

Proof. It suffices to show that for each RE-problem A and each associated RE-verifier V_A the pair (A, V_A) is gp-reducible to $(K', V_{K'})$. So let A be in RE, let V_A be an RE-verifier for A and let M be a Turing-machine that accepts V_A. We define a new machine M' whose work on the input x is described in pseudo-code as follows. The machine, that simulates the work of the machine M for V_A, uses a new special state h.

1. $y := \epsilon$ (ϵ is the empty word)
2. while $1 \neq 2$ do
3. begin
4. run $M(x, y)$
5. if $M(x, y)$ accepts
6. go to state h
7. set y to the quasi-lexicographic successor of y
8. end

The state h is designed, such that the machine, when it steps to state h (which only happens in line 6), immediately returns to the state that initializes the execution of line 7. So, if the simulation of the computation $M(x, y)$ in line 4 shows that M accepts (x, y), then M' switches to the state h for one computational step (line 6) and immediately continues in line 7.

Now, let i be the number of that machine M', i.e., $M' = M_i$. We define $f(x) := (i, x, h)$, for all $x \in \Sigma^*$. To see that $x \in A$ if and only if $f(x) \in K'$, assume that $x \in A$. Hence, there exists some $y' \in \Sigma^*$ such that V_A accepts (x, y'). In this situation, it is not hard to see that the machine M_i reaches the state h, when y is counted up to y'. It follows that $(i, x, h) \in K'$. Furthermore, in case of $x \notin A$ it is not hard to see that $M_i(x)$ never reaches h, that is $(i, x, h) \notin K'$.

Obviously, f is recursive. Hence, this reduction is a many-one reduction. We still have to show that it is also generating parsimonious.

It is easy to see that for each solution y' for x (w.r.t. V_A) the computation of $M_i(x)$ reaches h exactly once, i.e., there is exactly one solution k for $f(x) = (i, x, h)$ w.r.t. $V_{K'}$, a one-one correspondence between $V_A(x)$ and $V_{K'}((i, x, h))$. So, we define that the generating function g maps such a solution y' for x to the number k of the computational step, where the state h is reached after successfully testing $M(x, y')$. This number can be computed by simulating the work of the machine M_i and counting the computational steps until $M(x, y')$ is simulated and h is reached afterwards. Thus, g is recursive and we have $(A, V_A) \leq_{gp} (K', V_{K'})$ via f and g. □

By the definitions of $\exists_r\forall_l gp$-completeness and universal RE-verifiers, it immediately follows that $V_{K'}$ is universal for K'.

Corollary 5.2.2. *The* RE*-verifier $V_{K'}$ is universal for K'.*

The following theorem furthermore provides the RE-completeness for all problems of alternative solutions for K', which is needed to show hardness of all ASPs for some RE-problem via one *gp*-reduction from $(K', V_{K'})$.

Theorem 5.2.3. *For all $n \in \mathbb{N}^+$, $n + K'$ is* RE*-complete.*

Proof. The fact that $n + K' \in$ RE, for all $n \in \mathbb{N}$, is trivial. In order to show the RE-hardness, we will give many-one reductions $K' \leq_m n + K'$, for all $n \in \mathbb{N}^+$. So, let n be some natural number and let (i, x, j) be an arbitrary instance of K'. Starting with the machine M_i we design a machine M, as follows. The working alphabet of M includes the alphabet of M_i and the additional symbols # and \$. Using n new states, any computation of M starts with writing $\#^n\$$ to the left of the input x, producing the string $\#^n\$x$. Afterwards the machine M switches to the state j. When the machine reads a # in state j, it deletes it and moves right afterwards. If M reads a \$ while being in state j, it deletes it, moves right, and switches to the starting state of the original machine M_i. The work of the machine in state j on all symbols of the original alphabet of M_i is unchanged. So it is not hard to see that each computation starts with producing n #s and deletes them in n consecutive state-j steps. Let ℓ denote the number of the first state-j step. Thus, the n consecutive state-j steps have the numbers $\ell, \ell + 1, \ldots, \ell + n - 1$.

After this preprocessing, M works on x as M_i does. Let i' be the number of this machine, which is effectively computable. Hence, (i', x, j) has n trivial solutions $\ell, \ell + 1, \ldots, \ell + (n - 1)$ in the sense of K' and (i', x, j) has a further solution if and only if $(i, x, j) \in K'$. So $f((i, x, j)) := ((i', x, j), \ell, \ldots, \ell + (n - 1))$ realizes the reduction $K' \leq^p_m n + K'$. □

So, K' can play the role of the mentioned starting point for RE to prove $\exists_r\forall_l gp$-completeness, universality and the hardness of the problems of alternative solutions.

Theorem 5.2.4. *Let (B, V_B) be gp-reducible to (A, V_A) for a $\exists_r\forall_l gp$-complete problem B with an associated universal verifier V_B (e.g. K' with $V_{K'}$). Then A is also $\exists_r\forall_l gp$-complete, V_A is universal for A and $n + A$ (defined with respect to V_A) is* RE*-complete, for all $n \in \mathbb{N}^+$.*

Proof. The proof can by copied from Lemma 4.1.10. □

So, the situation seems to be the same as for NP. When we *gp*-reduce $(K', V_{K'})$ to (A, V_A) for an RE-problem A with an associated verifier V_A we answer the most important questions all at once. We show the $\exists_r\forall_l gp$-completeness of A via V_A, which implies that V_A is a universal verifier for A and we prove the RE-completeness of all alternative solution problems $n + A$, $n \in \mathbb{N}$ (Theorem 5.2.4). So we will try to *gp*-reduce $(K', V_{K'})$ to some RE-problems.

As a first candidate consider the halting problem K, which consists of all pairs $(i, x) \in \mathbb{N} \times \Sigma^*$ such that the machine M_i halts on the input x. Recall the natural RE-verifier V_K for K,

$$V_K = \{((i, x), k) : M_i(x) \text{ halts after exactly } k \text{ computational steps}\}.$$

Obviously, there is at most one solution k for each machine M_i w.r.t. V_K. Since there are instances for K' with more than one solution with respect to $V_{K'}$, $(K', V_{K'})$ can not be *gp*-reducible to (K, V_K).

Another possible (and still quite natural) RE-verifier for K would be

$$V'_K = \{((i, x), k) : M_i(x) \text{ halts after at most } k \text{ computational steps}\}.$$

Now, $V'_K(i, x) = \varnothing$ if $(i, x) \notin K$ and $V'_K(i, x) = \{k : k \geq k'\}$ if $M_i(x)$ halts after exactly k' computational steps. So each instance has no solutions or an infinite number of solutions w.r.t. V'_K. Since there are instances for K' with a finite and positive number of solutions with respect to $V_{K'}$, $(K', V_{K'})$ is not *gp*-reducible to (K, V_K).

However, the following theorem states that there exists a universal verifier V''_K for K.

Theorem 5.2.5. *There exists a verifier V''_K for K such that the pair $(K', V_{K'})$ is gp-reducible to (K, V''_K).*

Proof. For the proof we depart from the standard scheme of giving a verifier V''_K and then proving that $(K', V_{K'}) \leq_{gp} (K, V''_K)$ afterwards. We rather develop this verifier, while designing the *gp*-reduction $(K', V_{K'}) \leq_{gp} (K, V''_K)$.

So let (i, x, j) be an instance for K'. Recall that a solution for (i, x, j) (w.r.t. $V_{K'}$) is a number k such that the computation $M_i(x)$ is in state j after k computational steps. Our task is to map (i, x, j) to an instance (i', x') for the standard halting problem K. If j is the halting state of M_i, (i, x, j) is mapped to (i, x). Obviously, $M_i(x)$ reaches the state j if and only if $M_i(x)$ halts.

Let j be a state of M_i different from the halting state. In a first step, we modify the machine M_i to a machine M'_i such that

- the state j is the new halting state of M_i', i.e., $M_{i'}$ halts when M_i would reach the state j the first time and
- the old halting state of M_i is changed such that $M_{i'}$ runs into an endless loop, when it reaches this state.

It is easy to see, that $M_i(x)$ reaches the state j if and only if $M_{i'}(x)$ halts. The problem is that in case the computation of $M_i(x)$ reaches the state j several times, which means that (i, x, j) has several solutions w.r.t. $V_{K'}$, the machine $M_{i'}(x)$ halts only once, suggesting only one solution. So we have to adjust the notion of solutions (the verifier V_K'') for K such that the constructed instance (i', x) can have several solutions that correspond to the solutions of (i, x, j) w.r.t. $V_{K'}$.

The idea of this verifier V_K'' is to restart the machine M_i' after it has halted, simulating the actual behavior of the original machine M_i, that would not have halted. Such a restart must reflect, that a halting of the machine M_i' means that the machine M_i would have been in state j. To be able to do this restart properly, the program of the machine $M_{i'}$ must contain the information about how the machine M_i proceeds in state j. Since this information is lost by making j a halting state, we need some further adjustments of M_i'. We do so by adding a copy j' of the state j to M_i'. In case of M_i has states, different from the starting state, that do not appear on a right site of a transition rule of M_i', these states are needless and deleted in M_i'. Thus, this new state j' is the only one that does not appear on a right site of a transition rule of M_i' and the verifier V_K'' can easily identify this state j'. Thus, V_K'' can do the restart properly using this state j'. In case of j is the halting state of the original machine M_i, this information is also copied to j' and the verifier V_K'' does no restart.

Whenever the program of the verifier V_K'' notices that M_i' halts, that situation represents that M_i would have been in state j and the corresponding computational step is noticed as an solution of (i', x) w.r.t. to this just described verifier V_K''. Note that formally, given an input $((i, x), k)$, the algorithm for V_K'' starts the above simulating and restart process and counts its computational steps. It accepts when k is detected as solution as described above and rejects if the counter exceeds k. In the case that j is the halting state of M_i, the computation also rejects, if it stops before the counter reaches k.

So (i, x, j) is mapped to the pair (i', x) and the verifier V_K'' detects, for each solutions of (i, x, j) w.r.t. $V_{K'}$, a solution for (i', x). Hence, we have a $\leq_{gp}$-reduction $(K', V_{K'}) \leq_{gp} (K, V_K'')$.

It is very important to note that V_K'' actually is a verifier for K, since the whole restart process makes no sense for machines that are not designed by the above construction. However, V_K'' is indeed a verifier for K since it, for each input (i, x), detects a (first) solution if and only if $M_i(x)$ halts. □

By Theorem 5.2.5, it follows that the given verifier V_K'' is universal for K. But V_K'' is anything but naturally defined. We also do not expect that another more natural universal verifier for K exists. So, there are quite natural verifiers V_K

and V'_K for K that are not universal and the universal verifier V''_K for K is very unnatural. So the notion of universal verifiers and the intuition of natural verifiers diverge in the case of the halting problem K.

The following theorem implicitly states that the notions of natural and universal verifiers also differ for any other RE-problem.

Theorem 5.2.6. *Let A be a problem from* RE*. There exists an* RE*-verifier V'_A for A such that $|V'_A(x)| \leq 1$, for all $x \in \Sigma^*$.*

Proof. Let V_A be any RE-verifier for A. We define V'_A as follows,

$$V'_A = \{(x, y) : (x, y) \in V_A \text{ and } y \text{ is the lexicographic smallest solution for } x\}.$$

It is not hard to see that $V'_A = \varnothing$ if $x \notin A$ and $|V'_A| = 1$ if $x \in A$. Moreover, the following algorithm decides if $(x, y) \in V'_A$, using an algorithm for V_A:

1. if $(x, y) \notin V_A$, stop and reject the input (x, y)
2. $y' := \epsilon$ (ϵ is the empty word)
3. while $y' <_{lex} y$ do
4. begin
5. if $(x, y') \in V_A$, stop and reject the input (x, y)
6. set y' to the lexicographic successor of y'
7. end
8. accept the input (x, y)

So, V'_A is actually an RE-verifier for A such that $|V'_A(x)| \leq 1$, for all $x \in \Sigma^*$. □

Assume that V_A is a natural verifier for an RE-problem A. Then, the above construction leads to another verifier V'_A for A that also deserves to be called natural. Note that V'_A can not be universal for A. Hence, each RE-problem with some natural verifier V_A also has a natural verifier that can not be universal, namely V'_A.

Even though we could easily transfer the notions of alternative solutions problems, *gp*-reduction, $\exists_r \forall_l gp$-reduction, $\exists_r \forall_l gp$-completeness and universal verifiers from NP to RE there is a structural difference between NP and RE concerning their notions of solutions. This difference is stated in the above theorem, which is highly unlikely to also hold for NP. First, note that the construction of V'_A from V_A does not lead to an NP-verifier V'_A for all NP-verifier V_A. The problem is that testing all lexicographic predecessors (steps 2 to 7 of the above algorithm) can not be done in polynomial time. So the above proof can not be converted to NP. The second and the stronger argument is the following theorem that (almost) rules out an assertion like Theorem 5.2.6 for NP.

Theorem 5.2.7. *There exists a verifier V_A with $(\forall x \in \Sigma^*)[|V'_A(x)| \leq 1]$, for each* NP*-problem A, if and only if* UP = NP*.*

Proof. The theorem is immediate by the definition of UP. □

Hence, the existence of an NP-equivalent of Theorem 5.2.6 would imply UP = NP, which is widely believed to be wrong.

5.3 Conclusions

Even though RE seemed to be a very suitable class for applying our NP-theory, it turned out that it is not. We introduced the notions of verifiers, alternative solution problem, *gp*-reduction, universal verifiers and $\exists_r \forall_l gp$-reduction for RE. Analogous to the NP-case we showed that the alternative solution problems of a certain version of the halting problem are RE-complete and that the associated natural verifier is universal. However, the natural verifier of the standard halting problem turned out to be not universal and another, universal verifier for the standard halting problem is very unnatural. So in contrast to NP, the notion of natural verifiers and universal verifiers seem to diverge in RE. The reason for that behavior is the property stated in Theorem 5.2.6 that provides the existence of a quite natural verifier for each RE-problem such that all instances have at most one solution with respect to this verifier. Obviously, such natural verifiers can not be universal and discussing alternative solutions for such verifiers makes no sense at all.

Even though the investigation of alternative solutions for RE-problems did not result in a happy end, we feel that it was not useless. The used theory that works great for NP did not accord with RE. This exposed that there must be a structural difference between these classes concerning their solutions, which was finally identified.

Chapter 6

Alternative Solutions in the Polynomial-Time Hierarchy

As already mentioned in Chapter 5 the existence of a plausible notion of solutions is the precondition for discussing alternative solutions. Fortunately, there exists a characterization for the problems from the classes $\Sigma_i^p, i \geq 1$, that provides a notion of solutions and thus invites us to study the complexity of alternative solutions for these classes. Since the very similar characterization of the classes $\Pi_i^p, i \geq 1$, does not lead to such an intuitive concept of solutions and moreover such a notion is also not in sight for the classes $\Delta_i^p, i \geq 1$, we have to restrict ourselves to studying the classes $\Sigma_i^p, i \geq 1$.

6.1 Preliminaries

The polynomial-time hierarchy was introduced in [Sto76].

Definition 6.1.1 ([Sto76]). *The classes Σ_i^p, Π_i^p, and $\Delta_i^p, i \geq 1$, are recursively defined as follows:*

1. $\Sigma_0^p = \Pi_0^p = \Delta_0^p = \mathrm{P}$.
2. *a)* $\Sigma_{i+1}^p = \mathrm{NP}^{\Sigma_i^p}$.
   b) $\Pi_{i+1}^p = \mathrm{co}\Sigma_{i+1}^p$.
   c) $\Delta_{i+1}^p = \mathrm{P}^{\Sigma_i^p}$.

As announced, there is a characterization for the classes Σ_i^p $i \geq 1$, that provides a notion of solution. It is stated in the following theorem, which furthermore gives a characterization for the classes Π_i^p, $i \geq 1$.

Theorem 6.1.2. *[SM73] Let L be a language and let $i \in \mathbb{N}$.*

1. *$L \in \Sigma_i^p$ if and only if there exists a language B from Π_{i-1}^p and a polynomial p such that for all $x \in \Sigma^*$,*

$$x \in A \leftrightarrow (\exists y)\ [|y| \leq p(|x|) \wedge (x, y) \in B].$$

2. *$L \in \Pi_i^p$ if and only if there exists a language B from Σ_{i-1}^p and a polynomial p such that for all $x \in \Sigma^*$,*

$$x \in A \leftrightarrow (\forall y)\ [|y| \leq p(|x|) \rightarrow (x, y) \in B].$$

Note that since $\Sigma_1^p = \text{NP}$ and $\Pi_0^p = \text{P}$ this theorem is a generalization of Theorem 3.1.1. Let A be a language from Σ_i^p and let B and p be an associated language from Π_{i-1}^p and a polynomial in the sense of the above Theorem 6.1.2. Then, analogous to the situation for NP, we can say that a string y with $|y| \leq p(|x|)$ is a solution for the instance x if and only if $(x, y) \in B$. Unfortunately, the universal quantifier in the characterization of the classes Π_i^p does not support such a plausible notion of solutions for Π_i^p.

The mentioned idea of solutions for Σ_i^p-languages leads to the following concepts of Σ_i^p-verifiers and solutions.

Definition 6.1.3. *Let $i \geq 0$ be a natural number.*

1. *A Σ_i^p-*verifier *V is a language from Π_{i-1}^p such that there exists a polynomial p satisfying $(\forall x, y \in \Sigma^*)[(x, y) \in V \rightarrow |y| \leq p(|x|)]$.*

2. *The language $L(V)$ accepted by a Σ_i^p-verifier V is defined as $L(V) = \{x \in \Sigma^* : (\exists y \in \Sigma^*)[(x, y) \in V)]\}$.*

3. *For a Σ_i^p-verifier V and a string $x \in \Sigma^*$, $V(x)$ denotes the* set of solutions *for x, that is $V(x) = \{y \in \Sigma^* : (x, y) \in V\}$.*

Note that for $i = 1$ ($\Sigma_1^p = \text{NP}$) this definition coincides with Definition 3.1.2 for the polynomial-time verifier for NP given in Chapter 2.

Because of the strong analogy to NP, we can easily transfer the notions of *gp*-reduction, $\exists_r \forall_l gp$-reduction, $\exists_r \forall_l gp$-completeness, universal verifiers and the associated results from NP to Σ_i^p, $i \geq 1$. We forbear from repeating these definitions and assertions.

6.2 Σ_i^p-SAT and Π_i^p-SAT

In this section we deal with the problems Σ_i^p-SAT and Π_i^p-SAT which are probably the best-known problems of the polynomial-time hierarchy. We show that the problems of alternative solutions for Σ_i^p-SAT,$i \geq 1$, are as hard as Σ_i^p-SAT (Σ_i^p-complete), where Π_i^p-SAT is the basis for the verifier $V_{\Sigma_{i+1}^p\text{-SAT}}$ for Σ_{i+1}^p-SAT. Moreover, we show that Σ_i^p-SAT is $\exists_r \forall_l gp$-complete for Σ_i^p via the universal verifier $V_{\Sigma_{i+1}^p\text{-SAT}}$, $i \geq 1$, providing that Σ_i^p-SAT is able to be a starting point for Σ_i^p, as SAT was for NP.

For each natural number i, the instances of Σ_i^p-SAT and Π_i^p-SAT are Boolean formulas F over a variable set X with a fixed partition $X = X_1 \cup X_2 \cup \cdots \cup X_i$ into

pairwise disjoint sets $X_1, \ldots, X_i$. In the following β_k will denote an assignment for X_k, $1 \leq k \leq i$, and by writing $F(\beta_1, \beta_2, \ldots, \beta_i)$ we mean the truth value of $F(\beta)$, where $\beta(x) = \beta_k(x)$, for all $x \in X_k$, $1 \leq k \leq i$.

Such a formula F is a member of Σ_i^p-SAT if and only if

$$(\exists\beta_1)(\forall\beta_2)(\exists\beta_3)\ldots(Q\beta_i)\ [F(\beta_1, \beta_2, \ldots, \beta_i) = 1],$$

where the quantifier Q is $\exists(\forall)$ if i is odd (even). The formula F is in Π_i^p-SAT if and only if

$$(\forall\beta_1)(\exists\beta_2)(\forall\beta_3)\ldots(Q\beta_i)\ [F(\beta_1, \beta_2, \ldots, \beta_i) = 1],$$

where the quantifier Q is $\forall(\exists)$ if i is odd (even).

It is a well known fact that Σ_i^p-SAT is complete in Σ_i^p and Π_i^p-SAT is Π_i^p-complete.

Theorem 6.2.1 ([Wra76]). *For all $i \in \mathbb{N}^+$, Σ_i^p-SAT is Σ_i^p-complete and Π_i^p-SAT is Π_i^p-complete.*

To discuss (alternative) solutions of Σ_i^p-SAT, we need a Σ_i^p-verifier for Σ_i^p-SAT. The following verifier $V_{\Sigma_i^p\text{-SAT}}$ seems to be a natural choice. We will later see that $V_{\Sigma_i^p\text{-SAT}}$ is a universal verifier for Σ_i^p-SAT.

Lemma 6.2.2. *The language $V_{\Sigma_i^p\text{-SAT}}$ as defined by*

$$V_{\Sigma_i^p\text{-SAT}} = \{(F, \beta_1) : F(\beta_1) \in \Pi_{i-1}^p\text{-SAT}\}$$

is a Σ_i^p-verifier for Σ_i^p-SAT, where $F(\beta_1)$ arises from F by substituting each variable x from X_1 by $\beta_1(x)$.

Proof. Given F and β_1, $F(\beta_1)$ can easily be computed. Since Π_{i-1}^p-SAT is in Π_{i-1}^p, we have $V_{\Sigma_i^p\text{-SAT}} \in \Pi_{i-1}^p$. Furthermore it holds that

$$\begin{aligned} F \in \Sigma_i^p\text{-SAT} &\leftrightarrow (\exists\beta_1)[(\forall\beta_2)(\exists\beta_3)\ldots(Q\beta_i)\ [F(\beta_1, \beta_2, \ldots, \beta_i) = 1]] \\ &\leftrightarrow (\exists\beta_1)[F(\beta_1) \in \Pi_{i-1}^p\text{-SAT}] \\ &\leftrightarrow (\exists\beta_1)[(F, \beta_1) \in V_{\Sigma_i^p\text{-SAT}}], \end{aligned}$$

where Q is $\exists$ $(\forall)$ if i is odd (even). Hence $V_{\Sigma_i^p\text{-SAT}}$ is a Σ_i^p-verifier for Σ_i^p-SAT. □

Note that for $i = 1$ (Σ_1^p-SAT = SAT) β_1 is an assignment for the entire variable set $X = X_1$ of F. Hence, $F(\beta_1) \in \{0, 1\}$ and thus, $(F, \beta_1) \in V_{\Sigma_1^p\text{-SAT}}$ if and only if β_1 satisfies F. So, $V_{\Sigma_1^p\text{-SAT}} = V_{\text{SAT}}$.

Theorem 6.2.3. *For all $n \in \mathbb{N}^+$, $n + \Sigma_i^p$-SAT (defined with respect to $V_{\Sigma_i^p\text{-SAT}}$) is Σ_i^p-complete.*

Proof. It suffices to give a reduction Σ_i^p-SAT $\leq_{gp}^p 1 + \Sigma_i^p$-SAT. Let F be an instance for Σ_i^p-SAT, that is, a Boolean formula over a fixed partitioned variable set $X = X_1 \cup X_2 \cup \cdots \cup X_i$ with say $X_1 = \{x_1, x_2, \ldots, x_n\}$. We define F' to be the formula $F' := (F \wedge \overline{x_{n+1}}) \vee (x_1 \wedge x_2 \wedge \cdots \wedge x_n \wedge x_{n+1})$ over the variable set $X' = X_1' \cup X_2 \cup \cdots \cup X_i$, where $X_1' = X_1 \cup \{x_{n+1}\}$. Since F' is already satisfied if all x_k, $1 \leq k \leq n+1$, are set to 1, it is easy to see that the assignment $\alpha_1 : X_1' \mapsto \{0,1\}$ that maps all variables to 1 is a solution for F' in the sense of Σ_i^p-SAT. Furthermore, it is obvious that each alternative solution must assign 0 to x_{n+1}. It follows that all alternative solutions for F' are also solutions for the original formula F, if the assignment of the variable x_{n+1} is discarded. Conversely, it is not hard to see that each solution β_1 of F becomes a solution β_1' of F', by additionally assigning 0 to x_{n+1}.

Hence, by defining $f(F) := (F', \alpha_1)$, we have a bijection g between the solutions of F and the solutions of F' different from α_1, via $g(\beta_1) := \beta_1'$, where β_1' equals β on $x_1, x_2, \ldots, x_n$ and $\beta_1'(x_{n+1}) = 0$. Obviously, f and g are polynomial-time computable and thus we have Σ_i^p-SAT $\leq_{gp}^p 1 + \Sigma_i^p$-SAT via f and g. □

Having a closer look at the proof for the Σ_i^p-completeness of Σ_i^p-SAT [Wra76], we can extract the following stronger result.

Theorem 6.2.4. *For each natural number i, Σ_i^p-SAT is $\exists_r \forall_l gp$-complete for Σ_i^p via $V_{\Sigma_i^p\text{-SAT}}$.*

Proof. Basically, the proof follows the idea from [Wra76] but some adjustments are necessary.

Since we know that Σ_1^p-SAT = SAT is $\exists_r \forall_l gp$-complete for Σ_1^p = NP via $V_{\text{SAT}} = V_{\Sigma_1^p\text{-SAT}}$ (Theorem 3.3.10), we can assume $i \geq 2$ in the following.

We have to show that each problem from Σ_i^p with any associated verifier is gp-reducible to $(\Sigma_i^p\text{-SAT}, V_{\Sigma_i^p\text{-SAT}})$. So, let $A \in \Sigma_i^p$ with some Σ_i^p-verifier V_A for A. Let p_1 be a polynomial such that $(\forall x, y \in \Sigma^*)[(x,y) \in V_A \rightarrow |y| \leq p_1(|x|)]$. We design a new verifier $\widetilde{V}_A^{i-1} \in \Pi_{i-1}^p$ for A which is based on V_A using a new symbol $\#$ as follows,

$$\widetilde{V}_A^{i-1} = \{(x, y\#^k) : (x,y) \in V_A \wedge k + |y| = p_1(|x|)\}.$$

Hence, $|y| = p_1(|x|)$ for all x, y with $(x,y) \in \widetilde{V}_A^{i-1}$ and there exists a canonical bijection between solutions w.r.t V_A and solutions w.r.t. $\widetilde{V}_A^{i-1}$.

Now we can state

$$x \in A \leftrightarrow (\exists y_1 \text{ with } |y_1| = p_1(|x|))\ [(x, y_1) \in \widetilde{V}_A^{i-1}].$$

According to Theorem 6.1.2 we can express $\widetilde{V}_A^{i-1} \in \Pi_{i-1}^p$ with a language V_A^{i-2} from Σ_{i-2}^p and a polynomial p_2 as follows,

$$(x, y_1) \in \widetilde{V}_A^{i-1} \leftrightarrow (\forall y_2 \text{ with } |y_2| \leq p_2(|x|))\ [(x, y_1, y_2) \in V_A^{i-2}].$$

Even though a language from Σ_i^p, $i \in \mathbb{N}$, are no verifiers, for simplification, we will call such a languages V_A, e.g. V_A^{i-2}, a verifier and a string y with $(x, y) \in V_A$ a solution for x.

Analogously to $\widetilde{V}_A^{i-1}$ we design a verifier $\widetilde{V}_A^{i-2}$ from V_A^{i-2} with solutions of uniform length.

Below, we will only handle verifiers with uniform lengthes and thus, for the lengths of the variables y_k, $1 \leq k \leq i$, it will always hold $|y_k| = p_k(|x|)$. We will use the quantifiers $(\forall y_k)$, $(\exists y_k)$ to abbreviate the terms $(\forall y_k$ with $|y_k| = p_k(|x|))$ and $(\exists y_k$ with $|y_k| = p_k(|x|))$, respectively. Using this abbreviations, we obtain

$$x \in A \leftrightarrow (\exists y_1)(\forall y_2)[(x, y_1, y_2) \in \widetilde{V}_A^{i-2}],$$

where $|y_k| = p_k(|x|)$ for $k \in \{1, 2\}$. By iteratively using Theorem 6.1.2 and designing verifiers $\widetilde{V}_A^{i-3}, \ldots, \widetilde{V}_A^1$ with solutions of uniform lengths $p_3, \ldots, p_{i-1}$ respectively, we gain the expression

$$x \in A \leftrightarrow (\exists y_1)(\forall y_2) \ldots (Q y_{i-1})[(x, y_1, y_2, \ldots, y_{i-1}) \in \widetilde{V}_A^1],$$

where Q is $\forall$ $(\exists)$ and $\widetilde{V}_A^1$ is from $\Sigma_1^p = \text{NP}$ $(\Pi_1^p = \text{coNP})$ when i is odd (even).

Now we apply the idea of Cook [Coo71], who proved the NP-completeness of SAT by translating the work of a nondeterministic Turing machine on a given input into a Boolean formula.

First, let i be odd. Let M be a Turing machine for $\widetilde{V}_A^1 \in \text{NP}$ that exactly works in time $q(|z|)$, for all inputs z, where q is a polynomial. Note that the length of the input $z = (x, y_1, y_2, \ldots, y_{i-1})$ only depends on x since $|y_k| = p_k(|x|)$, $1 \leq k \leq i$. Let r denote the polynomial such that $|z| = r(|x|)$. The construction of Cook applied to M and an input $z = (x, y_1, y_2, \ldots, y_{i-1})$ provides a function f that maps such instances z to a formula F such that F is satisfiable if and only if $(x, y_1, y_2, \ldots, y_{i-1}) \in \widetilde{V}_A^1$. We just give the idea of this formula.

For each computational step s, $1 \leq s \leq q(r(|x|))$, for each cell c of the tape of M, and for each symbol a of the alphabet of M there is a variable $\alpha_{s,c,a}$ that expresses if the symbol a is in the cell c in step s. Furthermore, for each computational step s and state u of the machine, there is a variable $\alpha_{s,u}$ that expresses if the machine is in state u in step s. So, for each computational step of M, there is a set of variables that completely describes the current configuration. Thus, an assignment β of F describes the configuration in each computational step of M. In F, the variables for the starting configuration are fixed to values that represent the input $(x, y_1, y_2, \ldots, y_{i-1})$ on the input tape and the fact that the machine is in its starting state. Furthermore, the formula F makes sure that for all k, $1 \leq k < q(r(|x|))$, the configuration in step $k + 1$ is a valid successor of the configuration in step k according to the rules of M. Finally, the construction of F also ensures that in the last computational step $s = q(r(|x|))$ the machine is in an accepting state. Putting all parts together, it is not hard to see that F is satisfiable if and only if the machine M accepts the input $(x, y_1, y_2, \ldots, y_{i-1})$, that is, $(x, y_1, y_2, \ldots, y_{i-1}) \in \widetilde{V}_A^1$.

Note that the input $(x, y_1, \ldots, y_{i-1})$ only effects the description of the initial configuration $x\#y_1\#y_2\#\ldots\#y_{i-1}$ of M and the number of computational steps $q(r(|x|))$. In Cook's construction the variables according to the initial configuration are fixed to values that represent the given input $x\#y_1\#y_2\#\ldots\#y_{i-1}$ on the tape of M.

We slightly modify the construction such that the variables that describe the words $y_1, y_2, \ldots, y_{i-1}$ in the initial configuration are not fixed. Hence, the modified formula only depends on x. We call this formula F_x.

For a fixed x, the string y_k, $1 \leq k \leq i-1$, has the length $p_k(|x|)$. Hence, we exactly know which cells of the input tape, contain the string y_k, $1 \leq k \leq i-1$. Thus, we also know which variables describe these cells in Cook's formula F_x. Let X_k denote the set of variables that describes y_k, for $1 \leq k \leq i-1$. Then, the words $y_1, y_2, \ldots, y_{i-1}$ are represented by assignments $\beta_1, \ldots, \beta_{i-1}$ for $X_1, \ldots, X_{i-1}$.

We denote the remaining variables of F_x with X_i and obtain a partition of the variable set of F_x into $X_1, \ldots, X_i$. Let β_k denote an assignment of the variables from X_k. Now, we can state

$$x \in A \leftrightarrow (\exists\beta_1)(\forall\beta_2)\ldots(\forall\beta_{i-1})\ [(\exists\beta_i)\ [F_x(\beta_1, \beta_2, \ldots, \beta_{i-1}, \beta_i) = 1]].$$

Since the right hand side of the formula is equivalent to $F_x \in \Sigma_i^p$-SAT, we have

$$x \in A \leftrightarrow F_x \in \Sigma_i^p\text{-SAT}.$$

In case of i is even, the situation is quite similar. We start with the expression

$$x \in A \leftrightarrow (\exists y_1)(\forall y_2)\ldots(\exists y_{i-1})\ [(x, y_1, y_2, \ldots, y_{i-1}) \in \widetilde{V}_A^1],$$

where $|y_k| = p_k(|x|)$, for $1 \leq k \leq i-1$, and $\widetilde{V}_A^1$ is from $\Pi_1^p = \text{coNP}$. It holds that $\overline{\widetilde{V}_A^1} \in \text{NP}$. So, let M be a nondeterministic Turing machine that accepts $\overline{\widetilde{V}_A^1}$. Analogous to the case of an odd i, the machine M with an input $(x, y_1, y_2, \ldots, y_{i-1})$ can be converted to a formula F'_x (which is independent from $y_1, y_2, \ldots, y_{i-1}$) with certain variable sets $X_1, \ldots, X_{i-1}$ that correspond to $y_1, y_2, \ldots, y_{i-1}$. Let X_i denote the rest of the variables of F'_x. Now, we can state

$$\begin{aligned}
x \in A &\leftrightarrow (\exists y_1)(\forall y_2)\ldots(\exists y_{i-1})\ [(x, y_1, y_2, \ldots, y_{i-1}) \in \widetilde{V}_A^1] \\
&\leftrightarrow (\exists y_1)(\forall y_2)\ldots(\exists y_{i-1})\ [(x, y_1, y_2, \ldots, y_{i-1}) \notin \overline{\widetilde{V}_A^1}] \\
&\leftrightarrow (\exists\beta_1)(\forall\beta_2)\ldots(\exists\beta_{i-1})\ [(\forall\beta_i)\ [F'_x(\beta_1, \beta_2, \ldots, \beta_{i-1}, \beta_i) = 0]] \\
&\leftrightarrow (\exists\beta_1)(\forall\beta_2)\ldots(\exists\beta_{i-1})(\forall\beta_i)\ [\overline{F'_x}(\beta_1, \beta_2, \ldots, \beta_{i-1}, \beta_i) = 1] \\
&\leftrightarrow \overline{F'_x} \in \Sigma_i^p\text{-SAT}.
\end{aligned}$$

By defining $F_x := \overline{F'_x}$, for even i, we have $x \in A \leftrightarrow F_x \in \Sigma_i^p$-SAT, for all $i \geq 2$. So, we have $A \leq_m^p \Sigma_i^p$-SAT via the mapping $x \mapsto F_x$, for all $i \geq 2$. Furthermore, a solution y'_1 of x w.r.t. V_A canonically corresponds to the solution $y_1 = y'_1\#^i$

for x w.r.t. $\widetilde{V}_A^{i-1}$. It is also not hard to see that a solution y_1 for x corresponds to a solution β_1 of F_x w.r.t. Σ_i^p-SAT and vice versa, since β_1 is the assignment of the variables that encodes y_1. So with $g(y_i') := \beta_1$, we have a *gp*-reduction $(A, V_A) \leq_{gp}^p (\Sigma_i^p\text{-SAT}, \Pi_{i-1}^p\text{-SAT})$ via f and g. □

Note that the formula F from the proof of Cook's Theorem is actually a 3CNF-formula. Hence, the above proof also works for the following stronger theorem.

Theorem 6.2.5. *Let* Σ_i^p-3CNFSAT *(*Σ_i^p-3DNFSAT*) denote the restriction of* Σ_i^p-SAT *to Boolean formulas in* 3CNF *(*3DNF*). Let furthermore* $V_{\Sigma_i^p\text{-3CNFSAT}}$ *and* $V_{\Sigma_i^p\text{-3DNFSAT}}$ *denote the restrictions of* $V_{\Sigma_i^p\text{-SAT}}$ *to* 3CNF- *and* 3DNF*-formulas, respectively. It holds that*

1. Σ_i^p-3CNFSAT *is* $\exists_r\forall_l gp$*-complete for* Σ_i^p *via* $V_{\Sigma_i^p\text{-3CNFSAT}}$ *for all odd* i *and*
2. Σ_i^p-3DNFSAT *is* $\exists_r\forall_l gp$*-complete for* Σ_i^p *via* $V_{\Sigma_i^p\text{-3DNFSAT}}$ *for all even* i*.*

6.3 Some $\exists_r\forall_l gp$-Complete Problems in Σ_2^p

Analogous to SAT in the case of NP we have a starting point for showing the $\exists_r\forall_l gp$-completeness of Σ_i^p-problems, namely Σ_i^p-SAT. In this section we focus on Σ_2^p, since for the classes Σ_i^p, $i \geq 3$, only very few natural complete problems are known. We gather some natural Σ_2^p-complete problems that turn out to also be $\exists_r\forall_l gp$-complete. Thus, for these problems, we have the universality of the used verifiers and the Σ_2^p-hardness of all associated ASP's.

6.3.1 Generalized Subset Sum (GSS)

Problem Description (Generalized Subset Sum)

Given: Two vectors u and v of integers and an integer t.

Question: Does $(\exists x)(\forall y)[ux + vy \neq t]$ hold, where x and y are binary vectors having the same length as u and v, respectively ?

Standard Verifier:

$$V_{\text{GSS}} = \{((u, v, t), x) : \ u \in \mathbb{N}^n \text{ and } v \in \mathbb{N}^m \text{ for integers } n \text{ and } m,\ t \in \mathbb{N}, \text{ and } x \in \{0,1\}^n \text{ such that } ux + vy \neq t, \text{ for all } y \in \{0,1\}^m\}.$$

In [BKL+02] the authors give a function that converts a 3CNF-formula over n variables to a triple (u, v, t) of vectors u and v of length n and m, respectively, and an integer t such that for all $x \in \{0,1\}^n$,

$$F(x) = 1 \leftrightarrow (\exists y \in \{0,1\}^m)[ux + vy = t].$$

Furthermore, it is shown that by using this function and exploiting the coNP-completeness of $\overline{\text{3SAT}}$, each Σ_2^p-problem can be reduced to Generalized Subset Sum. It is also not hard to see that these reductions are *gp*-reductions, which implies the $\exists_r\forall_l gp$-completeness of GSS (via V_{GSS}) for Σ_2^p.

Theorem 6.3.1. GSS *is* $\exists_r\forall_l gp$*-complete for* Σ_2^p *via* V_{GSS}.

6.3.2 MinDNF

Problem Description (MinDNF)

Given: A DNF-formula F and an integer k.

Question: Is there a DNF-formula G of size $\leq k$[1], which is equivalent to F ?

Standard Verifier:

$$V_{\text{MinDNF}} = \{((F,k),G) : \ F \text{ is a DNF-formula, } k \in \mathbb{N}, G \text{ is a formula in DNF of size at most } k \text{ and } F \text{ and } G \text{ are equivalent}\}.$$

The Σ_2^p-completeness of MinDNF was shown in [Uma01]. There, the problem Σ_2^p-3DNFSAT is reduced to the problem Strongest Implicant Core (see also Section 6.3.4), which is strongly related to MinDNF. Afterwards a reduction SIC $\leq_m^p$ MinDNF is given.

The first reduction Σ_2^p-3DNFSAT $\leq_m^p$ SIC also works as a reduction to the restricted version $\text{SIC}_=$ of SIC, which is defined as follows.

Problem Description ($\text{SIC}_=$)

Given: A DNF-formula $F = t_1 \vee t_2 \vee \cdots \vee t_n$, where t_i are the monomials of F and an integer k.

Question: Is there an implicant $C \subseteq t_n$ of F of size exactly k ?

Standard Verifier:

$$V_{\text{SIC}_=} = \{((F,k),C) : \ F = t_1 \vee t_2 \vee \cdots \vee t_n \text{ is a DNF-formula, } k \in \mathbb{N}, \text{ and } C \subseteq t_n \text{ is an implicant of } F \text{ of size exactly } k\}.$$

It turns out that this reduction Σ_2^p-3DNFSAT $\leq_m^p$ $\text{SIC}_=$ is also a *gp*-reduction $(\Sigma_2^p\text{-3DNFSAT}, V_{\Sigma_2^p\text{-3DNFSAT}}) \leq_{gp}^p (\text{SIC}_=, V_{\text{SIC}_=})$. Furthermore, the given many-one reduction SIC $\leq_m^p$ MinDNF is a *gp*-reduction when restricted to $\text{SIC}_=$. It follows that we have a *gp*-reduction $(\Sigma_2^p\text{-3DNFSAT}, V_{\Sigma_2^p\text{-3DNFSAT}}) \leq_{gp}^p (\text{SIC}_=, V_{\text{SIC}_=})$ and a *gp*-reduction from $(\text{SIC}_=, V_{\text{SIC}_=})$ to $(\text{MinDNF}, V_{\text{MinDNF}})$, which implies the $\exists_r\forall_l gp$-completeness of MinDNF.

Theorem 6.3.2. MinDNF *is* $\exists_r\forall_l gp$*-complete for* Σ_2^p *via* V_{MinDNF}.

[1]Here, the size of a formula is the number of occurrences of literals in the formula.

6.3.3 Monotone Minimum Weight Word (MMWW)

For the description of the problem Monotone Minimum Weight Word we introduce the notions of Π_1-nondeterministic circuits and monotone sets. A Π_1-*nondeterministic circuit* is an ordinary Boolean circuit, whose input is partitioned into two bitstrings x and y. A Π_1-nondeterministic circuit C *accepts an input* x if and only if $(\forall y)\ [C(x,y) = 1]$. The number of *ones* in a bitstring x is denoted with $ones(x)$.

A *monotone set* is a subset $S \subseteq \{0,1\}^{|x|}$ for which $x \in S$ implies $x' \in S$, for all $x' \succeq x$, where $\succeq$ is the bitwise partial order on bitstrings.

Problem Description (Monotone Minimum Weight Word)

Given: A Π_1-nondeterministic circuit C accepting a monotone set and an integer k.

Question: Does C accept an input x with $ones(x) \leq k$?

Standard Verifier:

$$V_{\text{MMWW}} = \{((C,k),x) :\ C \text{ is a } \Pi_1 \text{ nondeterministic circuit accepting a monotone set, } ones(x) \leq k, \text{ and } C \text{ accepts } x\}.$$

The Σ_2^p-completeness of Monotone Minimum Weight Word was shown in [Uma99] by a reduction from Σ_2^p-SAT. Having a closer look at this reduction, it turns out that it is even generating parsimonious and thus, we have the following theorem.

Theorem 6.3.3. MMWW *is* $\exists_r \forall_l gp$*-complete for* Σ_2^p *via* V_{MMWW}.

6.3.4 Strongest Implicant Core (SIC)

The problem Strongest Implicant Core deals with implicants of Boolean formulas. An *implicant* of F is a conjunction C of literals for which $C \rightarrow F$. Recall that monomials of DNF-formulas and clauses of CNF-formulas are represented as sets of literals. We also represent implicants or conjunctions as sets of literals.

Problem Description (Strongest Implicant Core)

Given: A DNF-formula $F = t_1 \vee t_2 \vee \cdots \vee t_n$, where t_i are the monomials of F and an integer k.

Question: Is there an implicant $C \subseteq t_n$ of F of size at most k ?

Standard Verifier:

$$V_{\text{SIC}} = \{((F,k),C) :\ F = t_1 \vee t_2 \vee \cdots \vee t_n \text{ is a DNF-formula, } k \in \mathbb{N}, \text{ and } C \subseteq t_n \text{ is an implicant of } F \text{ of size at most } k\}.$$

As already mentioned in Section 6.3.2, in [Uma01], the author establishes a $\leq_m^p$-reduction Σ_2^p-3DNFSAT $\leq_m^p$ SIC, which can also be interpreted as *gp*-reduction (via the associated verifiers). Hence, we have the following theorem.

Theorem 6.3.4. SIC *is* $\exists_r\forall_l gp$*-complete for* Σ_2^p *via* V_{SIC}.

6.4 Conclusions

The situation in the polynomial-time hierarchy is very similar to the case of NP, where we conjectured that all many-one-hard problems are also $\exists_r\forall_l gp$-complete. Analogous to SAT $\in$ NP, we managed to show the $\exists_r\forall_l gp$-completeness of Σ_i^p-SAT for Σ_i^p and that the associated alternative solution problems are hard. In case of Σ_2^p we have shown the $\exists_r\forall_l gp$-completeness of several Σ_2^p-complete problems, which implies the following assertion.

Corollary 6.4.1. *The natural verifiers of the* Σ_2^p*-problems* Generalized Subset Sum, Strongest Implicant Core, MinDNF, Monotone Minimum Weight Word, *and* Σ_2^p-SAT *are universal and the associated alternative solution problems are* Σ_2^p*-complete.*

Because of this and the structural similarity of NP and Σ_2^p, we conjecture that all Σ_2^p-complete problems are also $\exists_r\forall_l gp$-complete for Σ_2^p.

Conjecture 6.4.2. *All* Σ_2^p*-complete problems are* $\exists_r\forall_l gp$*-complete via some universal verifier.*

Note that we did not manage to show $\exists_r\forall_l gp$-completeness for all of the considered Σ_2^p-complete problems. In particular the $\exists_r\forall_l gp$-completeness of Generalized 3CNF Consistency and Circuit Restriction remains open. However, we do believe that these $\exists_r\forall_l gp$-completeness results can be shown using some clever ideas we failed to have.

Open Problem 4. *Give* $\exists_r\forall_l gp$*-completeness proofs for* Generalized 3CNF Consistency *and* Circuit Restriction*!*

Because of the small number of natural problems, known to be complete in the classes Σ_i^p, $i \geq 3$, we did not explicitly study these classes. However, we have at least two $\exists_r\forall_l gp$-complete problems in each, namely Σ_i^p-SAT and Σ_i^p-3CNFSAT if i is odd, and Σ_i^p-3DNFSAT if i is even. Hence, it is reasonable to also believe in the analogous conjecture, for $i \geq 3$.

Open Problem 5. *Give arguments for or against a version of Conjecture 6.4.2 for* Σ_i^p*,* $i \geq 3$*!*

Chapter 7

Approximation of Alternative Solutions

All previous chapters dealt with the alternative solution problems for problems from NP, Σ_i^p, $i \geq 1$, or RE as decision problems. Except in case of RE, where the alternative solution problems turned out to be more or less senseless, we found out that in most of all cases the alternative solution problem of a problem A is as hard as A itself. Thus, an effective algorithm for computing alternative solutions of many NP-complete or Σ_i^p-complete problems can not exist, unless P = NP or the polynomial hierarchy collapses.

In this section we, instead of exactly computing an alternative solution, focus on approximating an alternative, or more precisely, a second best solution using the known optimal solution. By discussing several NP-optimization problems we will see that, compared to the decision problems, the situation is not so homogenous. For some problems there exists an approximation algorithm for the second best solution, way better than the approximability of the basis problem, e.g., MaxCut, MaxSat, and Minimum Steiner Tree. On the other hand, there are also problems where approximating a second best solution is as hard as approximating a first solution, e.g., MinTSP and MinIndDomSet.

7.1 Preliminaries

7.1.1 A short introduction to approximation algorithms

Here we give an introduction to approximation algorithms. The main source of this section is the compendium of NP-optimization problems from Viggo Kann [CK97]. If not otherwise mentioned, all given definitions and assertions can be found there.

Approximation algorithms are defined for optimization problems. Since the class of our main interest is NP, we focus on NP-optimization problems.

Definition 7.1.1. *An* NP*-optimization problem A is a 4-tuple $(I, sol, cost, goal)$ such that the following holds.*

1. $I \in \mathrm{P}$ *is the set of instances.*

2. *for an instance* $x \in I$, $sol(x)$ *denotes the feasible solutions of* x. *There exists a polynomial* p_A *such that* $(\forall y \in sol(x))[|y| \leq p_A(|x|)]$. *Moreover, for any* x, y *such that* $|y| \leq p_A(|x|)$, *it is decidable in polynomial time, whether* $y \in sol(x)$.

3. *For an instance* x *with a feasible solution* y, $cost(x, y) \in \mathbb{N}$ *denotes the costs of* y. *The function cost is in* FP *and is also called the objective function.*

4. $goal \in \{min, max\}$.

The class NPO *is the class of all* NP*-optimization problems.*

The goal of an NPO-problem is to find an optimum solution for a given instance x, that is, a feasible solution y such that

$$cost(x, y) = goal\{cost(x, y') \ : \ y' \in sol(x)\}.$$

In the following, *opt* will denote the function mapping an instance x to the cost of an optimum solution.

It is easy to see that if an NPO-problem can be solved in polynomial time, then its corresponding decision problem is also in P. As a consequence, if P $\neq$ NP, any NPO-problem with an associated NP-complete decision problem is not solvable in polynomial time. Hence, for those problems, it is reasonable to look for approximate solutions computable in polynomial time.

Definition 7.1.2. *Let* A *be an NPO-problem. Given an instance* x *and a feasible solution* y *of* x, *we define the approximation ratio of* y *with respect to* x *as*

$$R_A(x, y) = \max\left\{\frac{cost(x, y)}{opt(x)}, \frac{opt(x)}{cost(x, y)}\right\}.$$

Hence, the approximation ratio is always a number greater than or equal to 1 and the smaller approximation ratio, the better the solution.

Definition 7.1.3. *An NPO-problem* A *belongs to the class* APX *if and only if there exists some constant* $c > 1$ *and an algorithm* T *that for each input* $x \in I_A$ *returns a feasible solution* $T(x)$ *such that*

$$R_A(x, T(x)) \leq c.$$

In this situation we call T *a* c*-approximation for* A.

From a practical point of view, it is usually sufficient to have c-approximation problem with a conveniently small $c > 1$. Some problems allow for even better algorithms, so called approximation schemes.

Definition 7.1.4. *Let A be an NPO-problem. An algorithm T is said to be an approximation scheme for A if and only if, for any instance x of A and for any rational $\varepsilon > 1$, $T(x, \varepsilon)$ returns a feasible solution of x whose approximation ratio is at most ε.*

Optimization problems with approximation schemes are classified with respect to the running time of the approximation scheme.

Definition 7.1.5. *An NPO-problem A belongs to the class* PTAS *if and only if it admits a polynomial-time approximation scheme (*ptas*), that is, an approximation scheme whose running time is bounded by a polynomial $q_\varepsilon(|x|)$, for each $\varepsilon > 0$.*

Observe that the time complexity of an approximation scheme in the above definition may be of the type $2^{1/(\varepsilon-1)}p(|x|)$ or $|x|^{1/(\varepsilon-1)}$, where p is a polynomial. Thus, computations with ε values very close to 1 may be practically unfeasible. This leads us to the following definition.

Definition 7.1.6. *An NPO-problem A belongs to the class* FPTAS *if and only if it admits a fully polynomial-time approximation scheme (*fptas*), that is, an approximation scheme whose time complexity is bounded by $q(|x|, (\varepsilon-1)^{-1})$, where q is a polynomial.*

Clearly, the following inclusions hold:

$$\text{FPTAS} \subseteq \text{PTAS} \subseteq \text{APX} \subseteq \text{NPO}.$$

These inclusions are strict under the assumption that $\text{P} \neq \text{NP}$.

In order to introduce a notion of completeness in NPO and APX, we give the following notion of an approximation preserving reduction.

Definition 7.1.7. *Let A and B be two NPO-problems. A is said to be* ptas*-reducible to B, in symbols $A \leq_{ptas} B$, if three functions f, g, and c exist such that:*

1. *For any $x \in I_A$ and for any rational $\varepsilon > 1$, $f(x, \varepsilon) \in I_B$ is computable in time polynomial with respect to $|x|$.*
2. *For any $x \in I_A$, for any $y \in sol_B(f(x, \varepsilon))$, and for any rational $\varepsilon > 1$, $g(x, y, \varepsilon) \in sol_A(x)$ is computable in time polynomial with respect to both $|x|$ and $|y|$.*
3. *$c : \{q \in \mathbb{Q} : q > 1\} \to \{q \in \mathbb{Q} : q > 1\}$ is computable and invertible.*
4. *For any $x \in I_A$, for any $y \in sol_B(f(x, \varepsilon))$, and for any rational $\varepsilon > 1$,*

$$R_B(f(x, \varepsilon), y) \leq c(\varepsilon) \text{ implies } R_A(x, g(x, y, \varepsilon)) \leq \varepsilon.$$

The following theorem is a corollary of Definition 7.1.7.

Theorem 7.1.8. *If $A \leq_{ptas} B$ and $B \in$ APX (respectively, $B \in$ PTAS), then $A \in$ APX (respectively, $A \in$ PTAS).*

Definition 7.1.9. *An* NPO*-problem A is called* NPO*-complete if and only if, for each $B \in$ NPO, $B \leq_{ptas} A$.*

Definition 7.1.10. *An* NPO*-problem A is called* APX*-hard if and only if, for each $B \in$ APX, $B \leq_{ptas} A$. An* APX*-hard problem is* APX*-complete if it belongs to* APX*.*

By Theorem 7.1.8, it is immediate that if an NPO-problem A is NPO-complete (APX-hard) then $A \notin$ APX ($A \notin$ PTAS).

We will also make use of the following new type of a fptas-preserving reduction. In particular, we will use this notion of reduction to show that the problem of approximating the second best Steiner tree admits no fptas.

Definition 7.1.11. *Let A and B be two NPO-problems. A is said to be $\leq_{fptas}$-reducible to B if three functions f, g, and c exist such that:*

1. *For any $x \in I_A$, $f(x) \in I_B$ is computable in time polynomial with respect to $|x|$.*
2. *For any $x \in I_A$, for any $y \in sol_B(f(x))$, $g(x, y) \in sol_A(x)$ is computable in time polynomial with respect to both $|x|$ and $|y|$.*
3. *$c : \Sigma^* \times \{q \in \mathbb{Q} : q > 1\} \to \{c \in \mathbb{Q} : q > 1\}$ is computable in polynomial-time and $\frac{1}{|x|, p((\varepsilon-1)^{-1})} \leq c(x, \varepsilon) - 1$, for some polynomial p, for all $\varepsilon \in \{q \in \mathbb{Q} : q > 1\}$, $x \in I_A$.*
4. *For any $x \in I_A$, for any $y \in sol_B(f(x)$, and for any rational $\varepsilon > 1$,*

$$R_B(f(x), y) \leq c(x, \varepsilon) \text{ implies } R_A(x, g(x, y)) \leq \varepsilon.$$

This type of reduction preserves fptas' in the following way.

Theorem 7.1.12. *If $A \leq_{fptas} B$ and $B \in$ FPTAS, then $A \in$ FPTAS.*

Proof. We will give an fptas for A based on the $\leq_{fptas}$-reduction, given by functions f, g, c, and the fptas for B. Let $x \in I_A$ and $\varepsilon \in \{q \in \mathbb{Q} : q > 1\}$. First we compute $f(x)$. Then, we apply the ptas for A to the instance $(f(x), c(x, \varepsilon))$. Its running-time is polynomial in $1/(c(x, \varepsilon) - 1) \leq p(|x|, 1/(\varepsilon - 1))$ and $f(x)$ and thus, polynomial in $1/(\varepsilon - 1)$ and $|x|$. So we obtain a $c(\varepsilon, x)$-approximation y for $f(x)$ and use g to compute the ε-approximation $g(x, y)$ for x. Since the running-time of the whole process is polynomial in $1/(\varepsilon - 1)$ and $|x|$, the described algorithm is an fptas for A. □

7.1.2 The approximation problem for alternative solutions

Here, we formalize the problem, given an instance with an optimal solution, to approximate an alternative solution or the second best solution for NPO-problems.

Definition 7.1.13. *Let $A = (I, sol, cost, goal)$ be an* NPO*-problem. The problem of approximating the second best solution for A is*

Problem Description (SB-**A)**

Instance: *A pair (x, y_{opt}), where $x \in I$ and $y_{opt} \in sol(x)$ with $cost(y_{opt}) = opt(x)$.*

Feasible Solution: *A string y from $sol(x) \setminus \{y_{opt}\}$.*

Costs: *$cost(y)$.*

Goal: *goal.*

We call this problem SB-*A.*

Note that SB-A is not always an NPO-problem, because for an instance (x, y) deciding whether y is an optimal solution y_{opt} for x can usually not be done in polynomial-time. However, we imagine that an algorithm for SB-A is applied after a costly computation of y_{opt} and thus, assume that all inputs have the form (x, y_{opt}). Hence, we will treat SB-A as an NPO-problem.

7.2 Approximability Results

In this section we gather approximability results for SB-A for some NPO-problems A. Observe, that for NPO-problems like, e.g., Minimum Vertex Cover, Maximum Independent Set, Maximum Clique, Minimum Set Cover, Minimum Dominating Set, and many more, the optimal solution can be trivially modified to an alternative solution of cost $opt + 1$ or $opt - 1$. For instance, adding one arbitrary vertex to a given minimal vertex cover of a graph results in a vertex cover of size $opt + 1$. So, SB-A is not very interesting for such problems A.

Conversely, there are a number of problems A, for which showing that SB-A is as hard as A is almost trivial. For instance, consider the problem Longest Cycle. It is easy to see that finding (or approximating) the longest cycle in a graph G with n-vertices is the same as finding the second longest cycle in $G \cup C_{n+1}$ knowing the optimal solution C_{n+1}. Analogously, some more auto-reductions from Chapter 3 can be transferred to the approximation versions, e.g. for, Minimum Edge Cost Flow, and Maximum Knapsack.

So, a lot of NPO-problems have rather uninteresting second best version. However, there are some interesting problems and we discuss some of them in the following sections. The following table gives a summary of our results for several

SB-problems[1]. Note that we assume that P $\neq$ NP and thus, FPTAS $\subsetneqq$ PTAS $\subsetneqq$ APX $\subsetneqq$ NPO, in this chapter.

NPO-Problem A	Approximability	Approximability for SB-A
Cubic Longest Cycle	$\notin$ APX	$\in$ PTAS [BST99]
MaxCut	APX-complete	$\in$ PTAS
MaxSat	APX-complete	$\in$ PTAS
MinMaxMatching	APX-complete	abs error ≤ 1
Minimum Steiner Tree	APX-complete	$\in$ PTAS, $\notin$ FPTAS
Min-Δ-TSP	APX-complete	$\in$ PTAS
MinIndDomSet	NPO PB-complete	NPO PB-complete
MinTSP	NPO-complete	NPO-complete

Above, *abs error* $\leq k$ means that there exists a polynomial-time algorithm providing a feasible solution y with $|costs(y_{opt}) - costs(y)| \leq k$, where y_{opt} denotes the optimal solution.

The horizontal line separates the problems with easier SB-versions from the problems for which the optimal solution is useless for approximating the second best solution.

7.2.1 Maximum Cut

Problem Description (MaxCut)

Instance: A graph $G = (V, E)$.

Feasible Solution: A partition of V into disjoint sets V_1 and V_2.

Costs: The cardinality of the cut, i.e., the number of edges with one end point in V_1 and one endpoint in V_2.

Goal: max.

For MaxCut, there exists an excellent constant approximation algorithm with an approximation ratio of 1.14 [GW95]. Since MaxCut is APX-complete [PY91], MaxCut admits no ptas. In contrast, there exists a ptas for SB-MaxCut.

Theorem 7.2.1. SB-MaxCut *is in* PTAS.

Proof. Let $G = (V, E)$ be a graph with a maximum cut V_1, V_2 with cardinality $opt(G)$.

The algorithm for SB-MaxCut either moves one vertex from V_1 to V_2 or one vertex from V_2 to V_1 and selects this vertex such that the new cut is as large as possible. This can easily by done, by testing the size of the modified cuts for all possible choices to put a vertex $v \in V$ from V_1 to V_2 or from V_2 to V_1.

[1]The result for SB-Cubic Longest Cycle is from [BST99]

For the analysis, let k_v be the number of edges between V_1 and V_2 including v, for all vertices $v \in V$. Furthermore, for $v \in V_i$, $i \in \{1,2\}$, let ℓ_v denote the number of edges from v to some vertex in V_i. It is not hard to see that moving a vertex v from V_1 to V_2 or from V_2 to V_1 results in a cut of size $opt(G) - (k_v - \ell_v)$.

Furthermore, it holds that

$$\sum_{v \in V} k_v = 2 \cdot opt(G)$$

and thus, $min\{k_v : v \in V\} \leq 2 \cdot opt(G)/|V|$. It follows that $min\{(k_v - \ell_v) : v \in V\} \leq 2 \cdot opt(G)/|V|$ and hence, the above algorithm provides a solution of size at least $(1 - 2/|V|) \cdot opt(G)$, a $(1 + 2/(|V| - 2))$-approximation. This is a $(1+\varepsilon)$-approximation for $|V| \geq 2/\varepsilon + 2$. Thus, the algorithm is a ptas. □

We obtain the same result for Maximum Directed Cut which is also APX-complete [PY91], by applying the same idea.

7.2.2 Maximum Satisfiability

Problem Description (MaxSat)

Instance: A CNF-formula F over the variable set X.

Feasible Solution: An assignment for X.

Costs: The number of clauses satisfied by the truth assignment.

Goal: max.

In [PY91] it is shown that MaxSat is APX-complete. Thus, there is no ptas for MaxSat. But, we find a ptas for approximating the second best solution for MaxSat.

Theorem 7.2.2. SB-MaxSat *is in* PTAS.

Proof. Let F be a given CNF-formula over $X = \{x_1, \ldots, x_n\}$ with an optimal assignment α ($cost(F, \alpha) = opt(F)$). Let $\varepsilon > 1$. We discuss two cases. If $n \leq log(|F|)$, we check all 2^n assignments for X and thus, we find the exact second best solution in polynomial-time.

For the second case let $n > log(|F|)$. Let α_i denote the assignment that emerges from α by flipping the value for x_i. For each i, $1 \leq i \leq n$, let $d_i := opt(F) - cost(F, \alpha_i)$ denote the net loss of satisfied clauses by flipping the value of x_i. We output $\alpha' := \alpha_k$, where $d_k = min\{d_j : 1 \leq j \leq n\}$, as an approximation for the second best solution.

For the analysis, note that for each i, $d_i = l\ell_i - w_i$, where ℓ_i denotes the number of clauses C lost by flipping the value of x_i, i.e., $\alpha(C) = 1$ and $\alpha_i(C) = 0$, and w_i denotes the number of clauses C' won by flipping value of x_i, i.e., $\alpha(C') = 0$

and $\alpha_i(C') = 1$. Observe that while changing the value of x_i to obtain α_i from α, $1 \leq i \leq n$, each of the $opt(F)$ clauses, satisfied by α, is lost at most once such that

$$\ell_1 + \cdots + \ell_n \leq opt(F).$$

Hence, $min\{\ell_j : 1 \leq j \leq n\} \leq \frac{1}{n} \cdot opt(F)$. By $d_i \leq \ell_i$, $1 \leq i \leq n$, we have that

$$min\{d_j : 1 \leq j \leq n\} \leq min\{\ell_j : 1 \leq j \leq n\} \leq \frac{1}{n} \cdot opt(F),$$

that is, the net loss by flipping x_k is bounded by $\frac{1}{n} \cdot opt(F)$. Since, $\frac{1}{1-\frac{1}{n}} = 1 + \frac{1}{n-1}$, α' is a $(1 + \frac{1}{n-1})$-approximation for $opt(F)$ and thus also for the second best solution of F. Since $n > log(|F|)$, α' is an ε-approximation for sufficiently large $|F|$. It follows that the described algorithm is a ptas for SB-MaxSat. □

7.2.3 Minimum Independent Dominating Set

Problem Description (MinIndDomSet)

Instance: A graph $G = (V, E)$.

Feasible Solution: An independent dominating set (IDS) for G, i.e., a subset $V' \subseteq V$ such that for all $u \in V \backslash V'$ there exists a $v \in V'$ for which $\{u, v\} \in E$, and such that no two vertices in V' are joined by an edge from E.

Costs: The cardinality of the IDS, i.e., $|V'|$.

Goal: min.

The problem MinIndDomSet is not approximable within $|V|^{1-\varepsilon}$ for any $\varepsilon > 0$ [Hal93]. Furthermore, in [Kan94] MinIndDomSet was shown to be NPO PB-complete via so called linear reductions. The class NPO PB includes all NPO-problems whose objective function is polynomial bounded in the input size and a linear reduction is a $\leq_{ptas}$-reduction (f, g, c) with $c(\varepsilon) = 1 + c'(\varepsilon - 1)$ for some constant c'. It turns out that SB-MinIndDomSet is also not approximable within $|V|^{1-\varepsilon}$ and NPO PB-complete.

Theorem 7.2.3. SB-MinIndDomSet *is not approximable within* $|V|^{1-\varepsilon}$.

Proof. Assume to the contrary that A is a $|V|^{1-\varepsilon}$-approximation algorithm for SB-MinIndDomSet for some $\varepsilon > 0$. Using this assumption we will give a $|V|^{1-\varepsilon}$-approximation algorithm for MinIndDomSet.

So let $G = (V, E)$ be an instance for MinIndDomSet. We add a new vertex w to V and the edges $\{\{w, v\} : v \in V\}$ to E. Thus, $\{w\}$ is a minimal IDS of the resulting graph G'. It is not hard to see, that w is no member of any alternative independent dominating set. Furthermore, each IDS of G is also an IDS of G' and conversely, each IDS $V' \neq \{w\}$ of G' is also an IDS of G. Now, we apply algorithm

A to $(G', \{w\})$, which produces a $|V|^{1-\varepsilon}$-approximation V' for the second smallest IDS of G'. As seen above, this is also a $|V|^{1-\varepsilon}$-approximation for the minimal IDS of G, a contradiction.

Hence, the assumption is wrong which implies the theorem. □

Note that the above idea represents a linear reduction from MinIndDomSet to SB-MinIndDomSet. Thus, SB-MinIndDomSet is also NPO PB-complete.

7.2.4 Minimum Maximal Matching

Problem Description (MinMaxMatching)

Instance: A graph $G = (V, E)$.

Feasible Solution: A maximal matching E', that is, a subset $E' \subseteq E$ such that no two edges in E' share a common endpoint and every edge in $E \setminus E'$ shares a common endpoint with some edge in E'.

Costs: The cardinality of the matching E'.

Goal: min.

It was shown in [YG80] that MinMaxMatching is APX-complete. For the alternative solution problem we can give an algorithm computing a solution with a guarantied absolute error of at most 1.

Theorem 7.2.4. *There exists an algorithm for* SB-MinMaxMatching *providing a feasible solution with an absolute error less or equal than* 1 *in polynomial-time.*

Proof. Let $G = (V, E)$ be a graph with a minimum maximal matching M. In the following we describe an approximation algorithm for SB-MinMaxMatching.

First, compute the set V_1 of vertices incident to some edge of M and the set $V_2 := V \setminus V_1$. Since M is maximal, there are no edges joining vertices from V_2.

Now, assume that there exists an edge $\{v_1, v_2\}$ between $v_1 \in V_1$ and $v_2 \in V_2$. Note that M includes an edge $\{v_1, w\} \in M$ for some $w \in V_1$. Remove that edge from M and add $\{v_1, v_2\}$. If E includes an edge $\{w, v_2'\}$ for some v_2' from V_2, also add this edge. Then, we have a maximal matching M' of size $|M|$ or $|M| + 1$.

For the second case assume that there is no edge between V_1 and V_2. Thus all vertices from V_2 are isolated vertices and thus, irrelevant for matchings. From now on, we ignore these vertices and consider the graph $G' = (V_1, E)$. Note that M is a perfect matching for G'. Since M is a minimum maximal matching, all maximal matchings of G' are perfect.

If $E = M$, M is the only perfect matching for G' and thus, there is no feasible solution besides M. In this case, the algorithm outputs, that there is no feasible solution.

Alternatively, assume that there is an edge $\{v_1, w_1\}$ in $E \setminus M$. We add $\{v_1, w_1\}$to M and remove the two edges $\{v_1, v_2\}$ and $\{w_1, w_2\}$, incident to v_1 and w_1, from

M. Since this new matching can not be maximal, it follows that $\{x_2, y_2\} \in E$. By adding $\{x_2, y_2\}$ we obtain a second minimum maximal matching being an exact alternative solution.

In each case we have found a maximal matching of size at most $opt(G) + 1$ or the insight that no alternative maximal matching exists. □

7.2.5 Minimum Steiner Tree

Problem Description (MinST)

Instance: A complete graph $G = (V, E)$, edge weights $s : E \to \mathbb{N}$, and a subset $S \subseteq V$ of required vertices.

Feasible Solution: A Steiner tree T, i.e., a subtree of G that includes all the vertices in S.

Costs: The sum of the weights of the edges in the subtree T, i.e., $cost(T) = \sum_{e \in T} s(e)$.

Goal: min.

To the best of our knowledge the current best approximation algorithm for MinST is a 1.55-approximation [RZ00]. Furthermore, in [BP89] it was shown that MinST is APX-complete, even when restricted to edge weights from $\{1, 2\}$. Thus, there is no ptas for MinST. In case of SB-MinST we manage to prove the existence of a ptas.

Theorem 7.2.5. SB-MinST *is in* PTAS.

Proof. We give a ptas for SB-MinST. Let $\varepsilon > 1$ be a rational number. We give an ε-approximation algorithm with runtime p_ε for some polynomial p_ε. So let $((G, s, S), T_{opt})$ be an instance for SB-MinST, where $G = (V, E)$, $|V| = n$, and T_{opt} is an optimal Steiner tree for the MinST-instance (G, s, S). Let $E(T)$ denote the edges of a graph T, e.g., $E(T_{opt})$ are the edges of T_{opt}. We assume that the triangle inequality holds for s, since we can replace $s(\{v_1, v_3\})$ by $s(\{v_1, v_2\}) + s(\{v_2, v_3\})$ if $s(\{v_1, v_3\}) > s(\{v_1, v_2\}) + s(\{v_2, v_3\})$. In this case using the edge $\{v_1, v_3\}$ represents using $\{v_1, v_2\}$ and $\{v_2, v_3\}$. Furthermore, we can assume that there are no weightless edges in G since otherwise, the respective vertices can be united to a new vertex.

Let $T_{secbest}$ denote an (unknown) second best Steiner tree of (G, s, S), i.e., an optimal solution for $((G, s, S), T_{opt})$ as SB-MinST-instance. Let $E^*_{T_{secbest}}$ denote the edges from $T_{secbest} \setminus T_{opt}$, that share a vertex with an edge from T_{opt}, that is,

$$E^*_{T_{secbest}} = \{e \in E(T_{secbest}) \setminus E(T_{opt}) : (\exists e' \in E(T_{opt}))[|e \cap e'| = 1]\}.$$

Our algorithm consists of two different procedures called algorithm A and algorithm B. Depending on the number of edges in $E^*_{T_{secbest}}$, algorithm A or algorithm B outputs a sufficiently good approximation of $T_{secbest}$. Since $E^*_{T_{secbest}}$ is unknown, both algorithms are applied consecutively, both results are compared, and the better Steiner tree is output.

First, assume that $|E^*_{T_{secbest}}| \geq k := \lceil 1/(\varepsilon - 1) \rceil$. In this case the following easy algorithm A provides a sufficiently good approximation. Algorithm A adds the edge, that realizes the minimum $min\{s(e) : e \in E \setminus E(T_{opt}) \wedge (\exists e' \in E(T_{opt}))[|e \cap e'| = 1]\}$. In other words, A adds the smallest possible edge e, such that the resulting graph $T_{opt}+e$ is still connected. If adding e leads to a cycle, A removes the cycle edge with the biggest weight (of course, e is not removed). Note that all edges from $E^*_{T_{secbest}}$ have weight greater or equal the weight of e. Thus, $cost(T_{secbest}) \geq ks(e)$ and it holds

$$\begin{aligned} cost(T_{opt} + e) &= cost(T_{opt}) + s(e) \\ &\leq cost(T_{secbest}) + \frac{1}{k} cost(T_{secbest}) \\ &\leq \varepsilon cost(T_{secbest}). \end{aligned}$$

Thus, algorithm A is an ε-approximation if $|E^*_{T_{secbest}}| \geq k$. Obviously, algorithm A has a polynomial-runtime p_A.

If $|E^*_{T_{secbest}}| < k$, algorithm A must not be good enough. In this case algorithm B exactly computes $T_{secbest}$ or another second best solution of (G, s, S), which is obviously an ε-approximation. If (G, s, S) has several second best Steiner trees we use the notion $T_{secbest}$ very flexibly, that is, $T_{secbest}$ always denotes a second best Steiner tree that still fulfils all assumptions made w.l.o.g.

Algorithm B performs a smart exhaustive search over all Steiner trees T with $|E^*_T| < k$. In a first step B guesses a candidate E^* for $E^*_{T_{secbest}}$ (see also Figure 7.1). Since, $|E| \leq n^2$, there are at most $n^{2\ell}$ possibilities for $|E^*| = \ell$. Since

$$\sum_{\ell=1}^{k-1} (n^2)^\ell = \frac{(n^2)^k - 1}{n^2 - 1} \leq n^{2k},$$

n^{2k} is an upper bound for the number of possibilities in this step. Note that guessing means here, that all possibilities are consecutively generated and handed over to the next step of the algorithm.

Assume in the following that $E^*_{T_{secbest}}$ is guessed in the first step. The sought-after Steiner tree $T_{secbest}$ decomposes into $T_{secbest} \cap T_{opt}$ and $T_{secbest} \setminus T_{opt}$. In the second step, we guess all possibilities for $T_{secbest} \setminus T_{opt}$ using the fact that $E^*_{T_{secbest}}$ is the set of edges, where $T_{secbest}$ "leaves" T_{opt}. We know that (a) $T_{secbest} \setminus T_{opt}$ is a forest, where (b) the endpoints of the edges $E^*_{T_{secbest}}$ that are in T_{opt} are exactly the leaves of this forest, since (a) $T_{secbest}$ is a tree and (b) if there were further leaves in $T_{secbest} \setminus T_{opt}$, these leaves could be cut from $T_{secbest}$ to gain a better

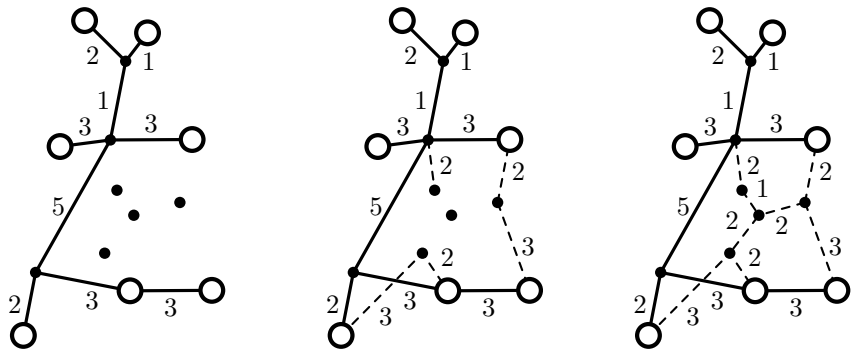

Figure 7.1: An illustration of the first and second step of algorithm B. (left) The optimal tree T_{opt}, (middle) the optimal tree T_{opt} plus the edges $E^*_{T_{secbest}}$ guessed in the first step, (right) the optimal tree T_{opt} plus the forest $T_{secbest} \setminus T_{opt}$ computed in the third step.

Steiner tree[2]. Furthermore, since the triangle inequality holds for s and since $T_{secbest} \setminus T_{opt}$ is an optimal forest (edge disjoint with T_{opt}) having these leaves, we can assume that this forest contains no vertices of degree 2. Because of that and since $|E^*_{T_{secbest}}| \leq k-1$, this forest contains at most $k-2$ further vertices. After guessing a set of at most $k-2$ vertices ($\leq n^{k-2}$ possibilities) the number of possibilities to form a forest from the given leaves and inner vertices is a number only depending on k, say $g(k)$. So for this step we have $\leq g(k)n^{k-2}$ possibilities (see also Figure 7.1).

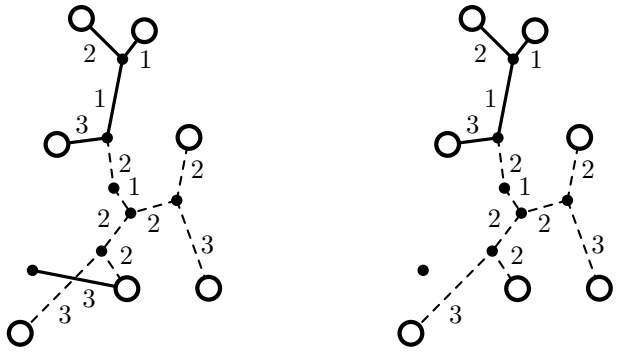

Figure 7.2: An illustration of the third step of algorithm B. (left) T_{opt} plus $T_{secbest} \setminus T_{opt}$ after subroutine C destroyed all cycles, (right) The second best solution $T_{secbest}$ obtained after deleting the useless edge.

Assume that $T_{secbest} \setminus T_{opt}$ was guessed in the second step and add it to T_{opt}. The resulting graph has several cycles. The number of cycles is bounded by $(k-1)^2$,

[2]Note that this is wrong in case of $T_{secbest}$ is T_{opt} plus an edge of small weight. However, in this situation algorithm A outputs $T_{secbest}$.

since T_{opt} and $T_{secbest} \setminus T_{opt}$ contain no cycles and there is at most one new cycle per path from a leave of $T_{secbest} \setminus T_{opt}$ to another. The number of such pathes is bounded by $(k-1)^2$. Now, the algorithm B has to find out, which is the best possible way to remove edges from T_{opt} such that the resulting graph is a tree and thus, $T_{secbest}$. This is done by a simple recursive subroutine C as follows. The routine C picks one cycle and consecutively works through all possibilities to delete one edge from the cycle and restarts C on the respective modified graph (see also Figure 7.2). After destroying all cycles algorithm B checks whether further edges (and possibly vertices) can be deleted without isolating a vertex from S. If this is possible, it is done as often as possible (see also Figure 7.2). During this process, the algorithm stores the recent best solution and finally deletes the according edges (and possibly vertices) from $T_{secbest} \cup T_{opt}$, resulting in $T_{secbest}$. An upper bound on the number of tested possibilities in this third step is n^{k^2} since, n is an upper bound for the number of edges in each cycle. Note that this subroutine C is not very clever and the estimate of the number of possibilities very rough. Both can surely be improved.

Let us take stock. Assume that $|E^*_{T_{secbest}}| < k$. Algorithm B guesses all $\leq n^{2k}$ possibilities for E^* with $|E^*| < k$ and thus also guesses $E^*_{T_{secbest}}$ in the first step. In the second step all possible forests connecting the chosen edges E^* are guessed. Here, for the correctly guessed $E^*_{T_{secbest}}$, we among others obtain $T_{secbest} \setminus T_{opt}$. In the second step there are $\leq g(k)n^{k-2}$ possibilities for some function g. The third step deals with finding the best possibility to remove T_{opt}-edges from $T_{secbest} \cup T_{opt}$ such that the resulting graph is a tree. Starting with the correctly guessed $T_{secbest} \cup T_{opt}$ this third step outputs $T_{secbest}$ after at most $n^{2(k-1)}$ tested possibilities. So, algorithm B tests overall $n^{2k} \cdot g(k)n^{k-2} \cdot n^{k^2} \leq g(k)n^{3k+k^2}$ possibilities which needs $\leq g(k)n^{3k+k^2} \cdot p(n)$ time for some polynomial p independent from k, since all but the guessing steps can be managed in polynomial time in n. In case of $|E^*_{T_{secbest}}| < k$, algorithm B finds and outputs $T_{secbest}$. Observe, that the running-time of algorithm B is a polynomial in n depending only from k, that is, from ε.

So executing algorithm A and algorithm B can be done in time p_ε for some polynomial p_ε and the better result is always an ε-approximation for $T_{secbest}$. □

Furthermore, we can show, that a ptas is the best possible approximation algorithm for SB-MinST, i.e., there is no fptas for SB-MinST.

Theorem 7.2.6. *There is no* fptas *for* SB-MinST.

Proof. We give a $\leq_{fptas}$-reduction from MinST to SB-MinST. Then, the theorem is immediate by Theorem 7.1.12. Let (G, s, S) be an instance for MinST, i.e., $G = (V, E)$ is a complete graph, $s : E \to \mathbb{N}$ is a weight function, and S is a subset of V. In a first step, we check if G has a Steiner tree of size zero and output this optimal Steiner tree if there exists one. This can be done in polynomial time by checking if the set of weight zero edges connects all vertices from S. So from now on, we assume that G' has no solution of weight zero. Let T_{opt} denote the

(unknown) optimal solution of the given instance (G, s, S). For this proof we use w (weight) as abbreviation for *cost*.

Let $S = \{s_1, \ldots, s_m\}$. The instance (G, s, S) is mapped to an instance (G', s', S') for SB-MinST as described in the following. First, we define a graph $\hat{G}$, that will be the basis of G'. The graph $\hat{G}$ consists of the vertices V, $\{s'_1, s''_1, \ldots, s'_m, s''_m\}$, $\{t_1, \ldots, t_m\}$, and a vertex d. The edge set $\hat{E}$ of $\hat{G}$ consists of E, $\{\{s_i, s'_i\} : 1 \leq i \leq m\}$, $\{\{s_i, s''_i\} : 1 \leq i \leq m\}$, $\{\{s''_i, t_i\} : 1 \leq i \leq m\}$, $\{\{s'_i, t_{i-1}\} : 2 \leq i \leq m\} \cup \{s'_1, t_m\}$, and $\{\{t_i, d\} : 1 \leq i \leq m\}$ (see also Figure 7.3).

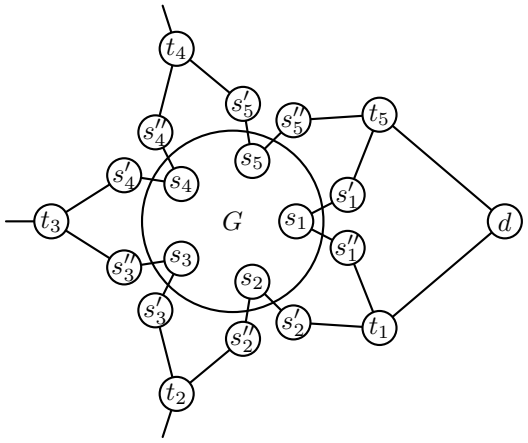

Figure 7.3: A depiction of the graph $\hat{G}$ based on G for $S = \{s_1, \ldots, s_5\}$. For simplicity, the graph G is only adumbrated by a cycle and only the beginnings of the edges $\{t_2, d\}$, $\{t_3, d\}$, and $\{t_4, d\}$ are drawn.

The graph G' is the complete graph over the vertex set of $\hat{G}$. The new weight function s' for G' is defined on the basis of $\hat{G}$. For the definition of the edge weights s', we furthermore need an approximation of the best Steiner tree of G. We compute such an approximation T_0 with weight $w(T_0)$ using the algorithm from [RZ00]. Note, that $w(T_0) \leq 1.55 w(T_{opt})$.

Now, the new weight function s' is defined as follows

$$s'(e) = \begin{cases} s(e), & \text{if } e \in E \\ w(T_0) + 1, & \text{if } e \in \hat{E} \text{ includes } s_i \text{ or } s'_i, 1 \leq i \leq m \\ 0, & \text{if } e \in \hat{E} \text{ includes } d \\ (2m+1)(w(T_0)+1), & \text{if } e \notin \hat{E}. \end{cases}$$

The set of required vertices is $S' = \{s'_1, s''_1, \ldots, s'_m, s''_m\}$. Obviously, each Steiner tree of the new instance must include one edge outgoing from each required vertex. Since each such edge has weight at least $w(T_0) + 1$, $2m \cdot (w(T_0) + 1)$ is a lower bound for the weight of the best Steiner tree. Thus, the tree using exactly the edges $\{\{s''_i, t_i\} : 1 \leq i \leq m\}$, $\{\{s'_i, t_{i-1}\} : 2 \leq i \leq m\} \cup \{s'_1, t_m\}$, and $\{\{t_i, d\} : 1 \leq i \leq m\}$

having weight $2m \cdot (w(T_0)+1)$ is optimal. We call this Steiner tree T'_{opt}. Note that we use primed letters to denote Steiner trees of G' and letters without primes for trees of G.

The idea of this construction is to ensure, that we have a known optimal solution T'_{opt} and that the second best solution corresponds to an optimal Steiner tree of the original instance. The following simple lemma helps to see, that this is actually the case.

Lemma 7.2.7. *Each Steiner tree of size* $< (2m+1)(w(T_0)+1)$ *includes either all the edges* $\{\{s''_i, t_i\} : 1 \leq i \leq m\}$, $\{\{s'_i, t_{i-1}\} : 2 \leq i \leq m\} \cup \{s'_1, t_m\}$, *or all the edges* $\{\{s_i, s'_i\} : 1 \leq i \leq m\}$ *and* $\{\{s_i, s''_i\} : 1 \leq i \leq m\}$.

Proof. Obviously, a Steiner tree T' of size $< (2m+1)(w(T_0)+1)$ includes at most $2m$ edges of weight $w(T_0)+1$ and no edges that are not included in $\hat{G}$. As seen above each Steiner tree must include one edge per required vertex. Thus, T' includes for each required vertex exactly one incident edge of weight $w(T_0)+1$. Now, it is not very hard to see that T' includes all edges of the first type or all edges of the second type, because, otherwise, T' is not connected and thus, no (Steiner) tree. □

As a first consequence, observe that T'_{opt} is the only Steiner tree of weight $< (2m+1)(w(T_0)+1)$ using the edges $\{\{s''_i, t_i\} : 1 \leq i \leq m\}$, $\{\{s'_i, t_{i-1}\} : 2 \leq i \leq m\} \cup \{s'_1, t_m\}$. Now, let $T' \neq T'_{opt}$ be a Steiner tree of (G', s', S') with $c(T') < (2m+1)(w(T_0)+1)$. Hence, T' includes all the edges $\{\{s_i, s'_i\} : 1 \leq i \leq m\}$ and $\{\{s_i, s''_i\} : 1 \leq i \leq m\}$ and it is not hard to see that the remaining edges span a Steiner tree $T_{T'}$ of the original instance (G, s, S) of weight $w(T') - 2m(w(T_0)+1) < w(T_0)+1$. Conversely, each Steiner tree T of G with $w(T) < w(T_0)+1$ induces a Steiner tree T'_T of G' with $w(T'_T) = 2m(w(T_0)+1) + w(T) < (2m+1)(w(T_0)+1)$.

By the initial computing of T_0, we have the existence of a Steiner tree of G of size $< w(T_0)+1$. Thus, it is not hard to see, that the second best Steiner tree $T'_{secbest}$ of G' induces the optimal Steiner tree of G, i.e., $T_{T'_{secbest}} = T_{opt}$.

We claim that the functions

- $f((G, s, S)) = (G', s', S')$,
- $g(G', T') = T_{T'}$, if $w(T') < (2m+1)(w(T_0)+1)$, and $g(G', T') = T_0$, otherwise, and
- $c((G, s, S), \varepsilon) = min\left\{1 + \frac{1}{2m(w(T_0)+1)+1.1w(T_0)}, 1 + \frac{\varepsilon - 1}{2.55 \cdot 2m+1}\right\}$

realize the required $\leq_{fptas}$-reduction from MinST to SB-MinST. All we still have to show is that $w(T') \leq c((G, s, S), \varepsilon) w(T'_{secbest})$ implies $w(g(G', T', \varepsilon)) \leq \varepsilon w(T_{opt})$.

First, we show that

$$w(T') \leq \left(1 + \frac{1}{2m(w(T_0)+1) + 1.1w(T_0)}\right) w(T'_{secbest})$$

implies that $w(T') < (2m+1)(w(T_0)+1)$ and thus, that T' induces a Steiner tree $T_{T'}$ of the original instance (G, s, S) as described above. In order to do so, assume that $w(T') \leq \left(1 + \frac{1}{2m(w(T_0)+1)+1.1w(T_0)}\right) w(T'_{secbest})$. It follows

$$\begin{aligned} w(T') &\leq \left(1 + \frac{1}{2m(w(T_0)+1)+1.1w(T_0)}\right)(2m(w(T_0)+1) + w(T_{opt})) \\ &\leq 2m(w(T_0)+1) + w(T_{opt}) + \frac{2m(w(T_0)+1) + w(T_{opt})}{2m(w(T_0)+1)+1.1w(T_0)}, \\ &< 2m(w(T_0)+1) + w(T_{opt}) + 1 \\ &\leq (2m+1)(w(T_0)+1). \end{aligned}$$

Now, let $w(T') \leq c((G, s, S), \varepsilon) w(T'_{secbest})$. Thus,

$$w(T') \leq \left(1 + \frac{1}{2m(w(T_0)+1)+1.1w(T_0)}\right) w(T'_{secbest})$$

which, as just seen, implies $g(G', T') = T_{T'}$. Note that we also have that

$$w(T') \leq \left(1 + \frac{\varepsilon - 1}{2.55 \cdot 2m + 1}\right) w(T'_{secbest}).$$

By $w(T') = 2m(w(T_0)+1) + w(T_{T'})$, it follows,

$$2m(w(T_0)+1) + w(T_{T'}) \leq \left(1 + \frac{\varepsilon - 1}{2.55 \cdot 2m + 1}\right)(2m(w(T_0)+1) + w(T_{opt})).$$

Hence,

$$\begin{aligned} w(T_{T'}) &\leq w(T_{opt}) + \frac{\varepsilon - 1}{2.55 \cdot 2m + 1}(2m(w(T_0)+1) + w(T_{opt})) \\ &\leq w(T_{opt}) + \frac{\varepsilon - 1}{2.55 \cdot 2m + 1}(2m(1.55w(T_{opt}) + w(T_{opt})) + w(T_{opt})) \\ &\leq w(T_{opt}) + \frac{\varepsilon - 1}{2.55 \cdot 2m + 1}(2m \cdot 2.55 + 1) w(T_{opt}) \\ &\leq \varepsilon w(T_{opt}). \end{aligned}$$

Thus, the given functions f, g, c realize the required reduction. □

7.2.6 Minimum Traveling Salesperson Problem

Problem Description (MinTSP)

Instance: A set C of n cities and a symmetric distance function $d : C \times C \to \mathbb{N}$.

Feasible Solution: A tour of C, i.e., a permutation $\pi : \{1..n\} \to \{1..n\}$.

Costs: The length of the tour.

Goal: min.

Note, that Traveling Salesman Problem-problems can also be defined as a graph problems, where the input is a complete graph with weighted edges. Then, a feasible solution is a Hamiltonian cycle. In this section we will use this equivalent characterization.

It is known that MinTSP is NPO-complete [OM90]. For SB-MinTSP, it turns out that a given optimal solution does not help in finding the second best solution, that is, SB-MinTSP is also NPO-complete.

Theorem 7.2.8. SB-MinTSP *is* NPO*-complete.*

Proof. For the proof we combine the argumentation for the NPO-completeness of MinTSP from [OM90], where Hamiltonian cycles play an important role, with an idea to generate an additional Hamiltonian cycle in a graph [PS76].

First, we give a sketch of the proof of the NPO-completeness of MinTSP from [OM90], that is, we give an idea of the $\leq_{ptas}$-reduction from the NPO-complete problem Weighted 3Satisfiability (see also [OM90]) to MinTSP.

Problem Description (Weighted 3Satisfiability)

Instance: A Boolean formula F in 3CNF over a variable set X and a weight function $w : X \to \mathbb{N}$.

Feasible Solution: All satisfying assignments α. The assignment $\alpha' : x \mapsto true$, $x \in X$, is always a feasible solution even if $F(\alpha') = false$.

Costs: The sum of the weights of the variables set to true, i.e., $cost(\alpha) = \sum_{\alpha(x)=1} w(x)$, for all satisfying assignments α, and $cost(\alpha') = 1 + \sum_{x \in X} w(x)$, if $F(\alpha') = 0$.

Goal: min.

This reduction is based on a well-known $\leq_m^p$-reduction from 3SAT to Hamiltonian Cycle from [PS82]. In this construction, a Boolean formula F over X is mapped to a graph G_F such that F is satisfiable if and only if G_F is Hamiltonian. Moreover, for each variable from $x \in X$, G_F includes an edge e_x, such that each Hamiltonian cycle C of G_F canonically corresponds to the satisfying assignment α_C, defined via $\alpha_C(x) = 1$ if $e_x \in C$ and $\alpha_C(x) = 0$, otherwise. Conversely, for each satisfying assignment α of F, G_F has a Hamiltonian cycle C_α with $\alpha_{C_\alpha} = \alpha$.

Based on this construction, the reduction Weighted 3Satisfiability $\leq_{ptas}$ MinTSP can be described as follows. Let (F, w) be an instance of Weighted 3Satisfiability, and let $G_F = (V, E)$ be the above graph. Let $G'_F = (V, E')$

be the complete graph over V. The distance function $d : E' \to \mathbb{N}$ is defined as follows,

$$d(e) = \begin{cases} 1 + \sum_{x \in X} w(x), & \text{if } e \in E' \setminus E \\ w(x), & \text{if } e = e_x, \text{ for some } x \in X \\ 0, & \text{otherwise.} \end{cases}$$

So, all edges not in G_F have a very large penalty weight $1 + \sum_{x \in X} w(x)$ and a Hamiltonian cycle C_α corresponding to a satisfying assignment α of F has weight $\sum_{\alpha(x)=1} w(x) = cost(\alpha)$, which is smaller than the penalty weight. Thus, it is not hard to see, that the functions f, g, and c defined via

- $f(((F, w), \varepsilon)) = (G'_F, d)$,
- $g((((G'_F, d), C, \varepsilon))) = \alpha_C$ if $cost((G'_F, d), C) \leq \sum_{x \in X} w(x)$, and $g(((x, C, \varepsilon))) = \alpha'$, otherwise, and
- $c(\varepsilon) = \varepsilon$

actually realize the reduction Weighted 3Satisfiability $\leq_{ptas}$ MinTSP.

We modify this reduction as follows. We make use of a construction given in [PS76], where, a graph G is mapped to a graph $\hat{G}$ with an additional Hamiltonian cycle C. In particular, $\hat{G}$ always has the Hamiltonian cycle C and it has an alternative Hamiltonian cycle for each Hamiltonian cycle of G. We apply this construction to the above graph G_F and call the resulting graph $\hat{G}_F = (\hat{V}_F, \hat{E}_F)$. Let C denote the additional cycle in $\hat{V}_F$.

Recall that Hamiltonian cycles of G_F correspond to satisfying assignments of F, and note that this correspondence is preserved by the construction of $\hat{G}_F$. In particular, there are still edges e_x, $x \in X$, in $\hat{G}_F$ such that each Hamiltonian cycle C', different from C, corresponds to a satisfying assignment $\alpha_{C'}$ of F as follows, $\alpha_{C'}(x) = 1$ if and only if $e_x \in C'$. It is important to note, that the cycle C contains none of the edges e_x, $x \in X$.

Now we can define the modified $\leq_{ptas}$-reduction. Let $\hat{G}'_F = (\hat{V}_F, \hat{E}'_F)$ be the complete graph over $\hat{V}_F$. Analogous to the case of Weighted 3Satisfiability $\leq_{ptas}$ MinTSP, the distance function $\hat{d} : \hat{E}'_F \to \mathbb{N}$ is defined as follows,

$$\hat{d}(e) = \begin{cases} 1 + \sum_{x \in X} w(x), & \text{if } e \in \hat{E}'_F \setminus \hat{E}_F \\ w(x), & \text{if } e = e_x, \text{ for some } x \in X \\ 0, & \text{otherwise.} \end{cases}$$

Again, edges not in $\hat{G}_F$ have large penalty weight. Since, C is a Hamiltonian cycle of $\hat{G}_F$ that uses non of the edges e_x, $x \in X$, it is the optimal solution and has weight zero. For each satisfying assignment α of F, $\hat{G}'_F$ has an alternative Hamiltonian cycle C'_α, having weight $\sum_{\alpha(x)=1} w(x) = cost(\alpha)$, which is again smaller than the penalty weight. Hence, the functions $\hat{f}$, $\hat{g}$, and $\hat{c}$ defined via

- $\hat{f}(((F, w), \varepsilon)) = ((\hat{G}'_F, d), C)$,
- $\hat{g}(((\hat{G}'_F, d), C), C', \varepsilon) = \alpha'_C$ if $cost(((\hat{G}'_F, d), C), C') \leq \sum_{x \in X} w(x)$, and $\hat{g}(((\hat{G}'_F, d), C), C', \varepsilon) = \alpha'$, otherwise, and
- $\hat{c}(\varepsilon) = \varepsilon$

realize the reduction Weighted 3Satisfiability $\leq_{ptas}$ SB−MinTSP. □

Note that using a very similar construction from [Krü05], where n additional Hamiltonian cycles are constructed, leads to the same result for approximating the nth best tour, given the best $n-1$ tours.

7.2.7 Minimum Metric Traveling Salesperson Problem

Problem Description (Min-Δ-TSP)

Instance: A set C of n cities and a symmetric distance function $d : C \times C \to \mathbb{N}$ satisfying the triangle inequality.

Feasible Solution: A tour of C, i.e., a permutation $\pi : \{1..n\} \to \{1..n\}$.

Costs: The length of the tour.

Goal: min.

The best known approximation algorithm for Min-Δ-TSP is the algorithm of Christofides [Chr76] having an approximation ratio of 1.5. Furthermore it is known, that there exists no ptas for Min-Δ-TSP, because Min-Δ-TSP is APX-complete [PY93]. For the problem of finding an alternative tour we have a better algorithm, namely a ptas.

Theorem 7.2.9. SB-Min-Δ-TSP *is in* PTAS.

Proof. Let C be a set of cities with a symmetric distance function d, that satisfies the triangle inequality, and let π denote a given minimal tour. The algorithm for SB-Min-Δ-TSP works as follows.

It scans π for the pair of consecutive cities c_1 and c_2 with minimal distance $d(c_1, c_2)$. Let c_0 denote the predecessor of c_1 in π and let c_3 denote the successor of c_2. The algorithm designs the new tour π' by replacing the part $c_0 \to c_1 \to c_2 \to c_3$ by $c_0 \to c_2 \to c_1 \to c_3$.

The costs of the modified tour π' are

$$cost(\pi') = cost(\pi) - d(c_0, c_1) - d(c_2, c_3) + d(c_0, c_2) + d(c_1, c_3).$$

By the triangle inequality, we have

$$\begin{aligned} cost(\pi') &\leq cost(\pi) - d(c_0, c_1) - d(c_2, c_3) + d(c_0, c_1) + d(c_1, c_2) \\ &\qquad + d(c_1, c_2) + d(c_2, c_3) \\ &\leq cost(\pi) + 2d(c_1, c_2). \end{aligned}$$

Since $d(c_1, c_2)$ is the smallest distance between two consecutive cities in the tour π, it is easy to see that $d(c_1, c_2) \leq cost(\pi)/n$. Hence, π' is a $(1+2/n)$-approximation for the optimal solution and thus, also for the second best solution. For sufficiently large C, it holds $2/n \leq \varepsilon$. Thus, the algorithm provides a ptas for SB-Min-Δ-TSP. □

7.2.8 Cubic Longest Cycle

The problem CHC plays an important role in this thesis, since it is the only treated problem, for which we, despite strong efforts, still do not know a universal verifier (see Section 4.4). The optimization version Cubic Longest Cycle of CHC is defined as follows.

Problem Description (Cubic Longest Cycle)

Instance: A cubic graph G.

Feasible Solution: A simple cycle C in G.

Costs: The length of the cycle.

Goal: max.

Probably because of the interesting properties of CHC concerning alternative solutions, there already are results for the approximability of Cubic Longest Cycle and in particular for SB-Cubic Longest Cycle. In [BST99] it was shown that Cubic Longest Cycle is not constant approximable. They also managed to show that there is a ptas for SB-Cubic Longest Cycle if the input graph is Hamiltonian. As seen above, the fact that SB-Cubic Longest Cycle is better approximable than Cubic Longest Cycle is in general not exceptional for NP-optimization problems and must not be linked to the seemingly extraordinary properties of CHC. However, we give a vague interpretation of the above results in the next section, again leading to the fact that CHC, respectively, Cubic Longest Cycle might be something special.

7.3 Conclusions

While discussing alternative solutions in terms of approximation, in contrast to the decision versions, it turns out that sometimes a given optimal solution is helpful in approximating alternative solutions and sometimes it is not. In particular we

have seen that the SB-Versions of the APX-complete (or even harder) optimization problems Cubic Hamiltonian Cycle, MaxCut, MaxSat, Minimum Steiner Tree, and Min-Δ-TSP admit a ptas. In case of MinMaxMatching the error bound of the best approximation algorithm drops from 2 to 1, when approximating the second best solution. On the other hand, the SB-versions of Minimum Independent Dominating Set and MinTSP are as hard as their basic versions. Naturally it would be interesting to analyze some more non-trivial alternative solution versions of NP-optimization problems.

Open Problem 6. *Study more* NP*-optimization problems!*

Even though we only considered a fistful of interesting problems, we want to give an interpretation of the results. Observe, that the APX-problems MaxCut, MaxSat, MinMaxMatching, Minimum Steiner Tree, and Min-Δ-TSP have easier SB-versions. In contrast, approximating the second best solution remains hard for the hard problems MinIndDomSet (NPO PB-complete) and MinTSP (NPO-complete). A possible justification is the following. If a problem is NPO- or NPO PB-hard, the properties of all NPO- or NPO PB-problems, respectively, can be encoded into instances of this problem, that is, a hard problem is very "flexible". We feel that, thus, it is more likely that a given instance of such a hard problem can be modified to have an additional optimal solution and to, nevertheless, represent the given instance and its solutions. Note that this is exactly what we did to show the hardness of SB-MinIndDomSet and SB-MinTSP.

Applying this "hard standard problem → hard SB-version"-rule to the problem Cubic Longest Cycle, one should rather expect that SB-Cubic Longest Cycle is hard, because Cubic Longest Cycle is not in APX. From this point of view, the fact that SB-Cubic Longest Cycle admits a ptas for Hamiltonian input graphs can be interpreted as an exception, once again motivating that CHC or Cubic Longest Cycle, respectively, actually has an exceptional position.

In contrast to the negative results for decision problems in the Chapters 3, 4, 5, and 6, we obtained some positive results for the approximability of alternative solutions, for instance for Minimum Steiner Tree and MaxSat. Thus, these results could be of practical relevance. However, from the practical point of view these positive results might have a drawback, since they only provide alternative solutions with small differences to the optimal solutions. In practice it is often necessary to have alternative solutions that substantially differ from the given optimal solutions. Thus, it would be an interesting task for future work, to find algorithms that provide alternative solutions with a large distance from the optimal solution, where the distance is formally expressed using some proper distance dimension.

Open Problem 7. *Study the approximability of second best solutions with a large distance from the given optimal solution!*

Chapter 8

Inverse-3DM is coNP-Complete

This chapter deals with inverse problems. The inverse problem Inverse-A, defined with respect to a verifier V_A for A is, given a set of solutions, to decide if there exists an instance having exactly those solutions [KS99, Che03]. The study of inverse problems does not directly contribute to the examination of alternative solutions even though an inverse problem with a candidate function can be formulated as an alternative solution problem. However, since solutions of NP-problems play a central role in inverse problems, we furthermore gain a better understanding of the solution structure of NP-problems. Note that inverse problems for the classes Σ_i^p, $i \in \mathbb{N}$, and RE have been studied in [Ber05].

The inverse problems of 3SAT, Clique, Hamiltonian Cycle, Partition, and Vertex Cover, defined with respect to their natural (and universal) verifiers, have been shown to be coNP-complete in [KS99, Che03, KH06]. Eying the list of the six basic NP-complete problems of Garey and Johnson [GJ79], the question of the inverse complexity of the last problem, 3Dimensional Matching, gains a special importance. In this chapter we close this gap, by also proving coNP-completeness for Inverse-3DM. This result has been published in [KH06].

Furthermore, we discuss the behavior of different universal verifiers for one and the same problem with respect to the associated inverse problems. In particular, we give strong arguments, that different universal verifier may induce inverse problems of different computational complexity.

8.1 Preliminaries

In this section we give the basic notions, needed to formulate our result, the coNP-completeness of Inverse-3DM. We give a short overview of the concept of inverse problems and introduce the involved problems Inverse-3DM and Inverse-3SAT and some helpful properties.

8.1.1 Inverse problems

As mentioned above, inverse problems deal with solutions of NP-problems and thus verifiers. We define the problem Inverse-(A, V_A).

Definition 8.1.1 ([Che03]). *The* inverse problem Inverse-(A, V_A) *for an* NP*-language A with a verifier V_A is the following problem.*

Given: *A set Π of potential solutions for A.*

Question: *Is there an instance x for A such that $V_A(x) = \Pi$?*

The inverse problem of a language $A \in$ NP can clearly only be defined relative to a verifier accepting A. For the treated problems 3SAT and 3DM we will use the natural verifiers V_{3SAT} and V_{3DM}, that have shown to be universal in Chapter 4. Note that the completeness results for the inverse problems of VC, Clique, Partition, and HC in [KS99, Che03, Krü05] have also been obtained using the corresponding natural (and universal, see Chapter 4) verifiers.

If it is clear, which verifier V_A is used for a given problem A, we denote the associated inverse problem with Inverse-A. Note that, when dealing with inverse problems it is more common to use the term *proofs* instead of solutions. So we will use this notion in the following.

A naive approach to solving the inverse problem Inverse-A (via V_A) would be, given a set of proofs Π, evaluate the proofs to compute a candidate string x and then check, if x satisfies $V_A(x) = \Pi$. For many natural verifiers for NP-complete problems such a candidate is relatively easy to compute.

Definition 8.1.2 ([Che03]). *Let V_A be a verifier for an* NP*-problem A. A polynomial-time computable mapping $c : \mathcal{P}(\Sigma^*) \to \Sigma^*$ is called a* candidate function for (A, V_A) *if and only if for all $\Pi \subseteq \Sigma^*$ the following holds: if there exists a $z \in \Sigma^*$ such that $V(z) = \Pi$ then $V(c(\Pi)) = \Pi$.*

It is not clear if all NP-problems with associated verifiers do have candidate functions. However, many natural verifiers for NP-complete languages, such as 3SAT or VC, have candidate functions. Note that if a problem A with V_A has a candidate function c then we have an obvious coNP upper bound for the complexity of Inverse-A, namely given Π, compute $c(\Pi)$, and then check if for all π such that $|\pi| \leq p(|c(\Pi)|)$ (where p is the polynomial that bounds the length of proofs with respect to the verifier V_A) we have $\pi \in \Pi \leftrightarrow V(c(\Pi), \pi) = 1$.

Observation 8.1.3. *Let $A \in$ NP with a verifier V_A. If there is a candidate function for A, then* Inverse-(A, V_A) is in coNP.

Let $A \in$ NP with a verifier V_A and let c_A be a candidate function for A. Hence, $\Pi \in$ Inverse-A if and only if $V_A(c_A(\Pi)) = \Pi$. Since $\Pi \subseteq V_A(c_A(\Pi))$ can be tested in polynomial time, the question if $\Pi \in$ Inverse-A reduces to the problem if $c_A(\Pi)$ has alternative solutions different from the ones in Π. Thus, for problems with a candidate function, the inverse problem is strongly related to the above alternative solution problem for candidates.

8.1.2 3Dimensional Matching

See Section 4.2.5 for a problem description.

In the following, we will use the shorthand Inverse-3DM for the language Inverse-$(3\text{DM}, V_{3\text{DM}})$.

Now, we want to gather some properties of 3DM and Inverse-3DM, that will finally lead to a candidate function $c_{3\text{DM}}$ for 3DM. First, assume that (S, X, Y, Z) is in 3DM. It is not hard to see, that thus, $X = X(S) := \{x : (\exists y)(\exists z)[(x, y, z) \in S]\}$, $Y = Y(S) := \{y : (\exists x)(\exists z)[(x, y, z) \in S]\}$, and $Z = Z(S) := \{z : (\exists x)(\exists y)[(x, y, z) \in S]\}$. Moreover, for each 3D-matching M of (S, X, Y, Z) also holds $X = X(M)$, $Y = Y(M)$, and $Z = Z(M)$.

Second, let $\Pi_{3\text{DM}} = \{M_1, M_2, \ldots, M_k\}$ be a set of (potential) proofs of $V_{3\text{DM}}$. If $\Pi_{3\text{DM}} \in$ Inverse-3DM then $\Pi_{3\text{DM}} = V_{3\text{DM}}((S, X, Y, Z))$ for a 4-tuple $(S, X, Y, Z) \in$ 3DM. Thus, we have $X = X(S)$, $Y = Y(S)$, $Z = Z(S)$, $|X(S)| = |Y(S)| = |Z(S)| = q$. For the matchings $M_1, M_2, \ldots, M_k$ furthermore holds, that $|M_i| = q, 1 \leq i \leq k$, and $X(M_i) = X$, $Y(M_i) = Y$, and $Z(M_i) = Y$, $1 \leq i \leq k$. Thus, a set of (potential) proofs $\Pi = \{M_1, M_2, \ldots, M_k\}$ can not be contained in Inverse-3DM if $|M_i| \neq |M_j|$, $X(M_i) \neq X(M_j)$, $Y(M_i) \neq Y(M_j)$, or $Z(M_i) \neq Z(M_j)$, for some $1 \leq i < j \leq k$. Following a more general concept from [Che03] we will hence call a set of (potential) proofs $\Pi = \{M_1, M_2, \ldots, M_k\}$ *well-formed* if and only if $M_1, M_2, \ldots, M_k$ include the same number q of triples and they all cover the same sets X, Y, and Z. Obviously, testing if a collection of sets is well-formed can be done in polynomial-time.

So a potential algorithm for Inverse-3DM might start with the test if the given set Π of proofs is well-formed. Afterwards, it is plausible to ask for a candidate in terms of the Definition 8.1.2 of candidate functions. Actually, such a candidate can be provided by the following candidate function.

Theorem 8.1.4. *The function $c_{3\text{DM}}$, defined via*

$$c_{3\text{DM}}(\Pi) := (\bigcup_{i=1}^{k} M_i, X(M_1), Y(M_1), Z(M_1)),$$

is a candidate function for $(3\text{DM}, V_{3\text{DM}})$.

Proof. Let $\Pi = \{M_1, M_2, \ldots, M_k\}$ be a given well-formed set of proofs (matchings). As seen above, for a candidate (S, X, Y, Z) for Π, it must hold that $X = X(M_1)$, $Y = Y(M_1)$, and $Z = Z(M_1)$ and thus $X(S) = X$, $Y(S) = Y$, and $Z(S) = Z$. Moreover, it is obvious, that S must contain all triples from all matchings $M_1, M_2, \ldots, M_k$. Now, it is not very hard to see, that the minimal instance with these properties is a candidate for Π, namely the instance $(\bigcup_{i=1}^{k} M_i, X(M_1), Y(M_1), Z(M_1)) = c_{3\text{DM}}(\Pi)$. □

Thus, by Observation 8.1.3 it follows that Inverse-3DM $\in$ coNP.

Corollary 8.1.5. *The problem* Inverse-3DM is in coNP.

8.1.3 3Satisfiability

One of the standard NP-complete problems is 3Satisfiability (3SAT, see Section 4.2.1 for a problem description). It plays a very important role in this chapter, since we will prove our main result, the coNP-completeness of Inverse-3DM by giving a many-one reduction from the inverse problem of 3SAT, which was shown to be coNP-complete in [KS99].

The inverse problem Inverse-$(3\text{SAT}, V_{3\text{SAT}})$ also has been denoted by Inverse-3SAT or 3SAT^{-1} [Che03, KS99]. Throughout this chapter we will use Inverse-3SAT to denote Inverse-$(3\text{SAT}, V_{3\text{SAT}})$.

As it has been the case for 3DM there are easy to check properties that any proof set for 3SAT must have in order to be in Inverse-3SAT. Since any assignment for an n-variable Boolean formula is represented by a string from $\{0,1\}^n$, the notion of well-formed proof sets Π with respect to $V_{3\text{SAT}}$ only requires that all strings from Π have the same length.

The following lemma from [Che03, KS99] provides a candidate function for $V_{3\text{SAT}}$.

Lemma 8.1.6 ([Che03, KS99]). *Let Π be a well-formed set of proofs for $V_{3\text{SAT}}$ and let n be the length of the strings in Π. Let the formula F_Π be the conjunction of all clauses C of exactly three literals over the variable set $\{x_1, \ldots, x_n\}$ such that C is satisfied by all assignments in Π.*

The mapping $c_{3\text{SAT}}$ that maps any well-formed proof set Π to the formula F_Π is a candidate function for $V_{3\text{SAT}}$.

It follows that Inverse-3DM $\in$ coNP. The coNP-hardness of Inverse-3SAT, which is important for the proof of our main result, was shown by Kavvadias and Sideri in [KS99].

Theorem 8.1.7 ([KS99]). Inverse-3SAT is coNP-complete.

Another concept that will be essential for our purposes was defined in [KS99], as well.

Definition 8.1.8 ([KS99]). *Let Π be a set of Boolean assignments for $x_1, \ldots, x_n$.*

1. *An assignment α for $x_1, \ldots, x_n$ is said to be* $\{x_i, x_j, x_k\}$-compatible *with Π, $1 \leq i < j < k \leq n$, if and only if there exists an assignment $\beta \in \Pi$ such that α and β assign the same truth values to x_i, x_j, and x_k.*

2. *An assignment α for $x_1, \ldots, x_n$ is called* 3-compatible *with Π if and only if it is $\{x_i, x_j, x_k\}$-compatible with Π for each triplet $\{x_i, x_j, x_k\}$ of variables, $1 \leq i < j < k \leq n$.*

Note that any assignment from Π is trivially 3-compatible with Π.

The following theorem gives a characterization of Inverse-3SAT that is based on the notion of 3-compatibility. It will play an important role in the proof of the upcoming main result.

Theorem 8.1.9 ([KS99]). *A well-formed set of proofs* Π *for* $V_{3\text{SAT}}$ *is in* Inverse-3SAT if and only if it is closed under 3-compatibility, i.e., if and only if for each assignment α that is 3-compatible with Π holds $\alpha \in \Pi$.

8.2 Inverse-3DM is coNP-complete

In this section we show the coNP-completeness of Inverse-3DM. The main idea is similar to the proof of the coNP-completeness of Inverse-HC in [KH06], i.e., we give a many-one reduction from Inverse-3SAT.

Theorem 8.2.1. Inverse-3DM is coNP-complete.

Proof. The fact that Inverse-3DM $\in$ coNP is stated in Corollary 8.1.5. We will prove coNP-hardness of Inverse-3DM by giving a $\leq_m^p$-reduction from Inverse-3SAT that was shown to be coNP-complete in [KS99] (see Theorem 8.1.7). This reduction will be done by the function f that will be defined during the proof. Let $\Pi_{3\text{SAT}}$ be a set of proofs for $V_{3\text{SAT}}$. If $\Pi_{3\text{SAT}}$ is not well-formed we define $f(\Pi_{3\text{SAT}})$ to be a fixed non-member of Inverse-3DM.

So let $\Pi_{3\text{SAT}}$ be a well-formed set of proofs. Hence, there exist natural numbers n and m such that $\Pi_{3\text{SAT}} = \{\alpha_1, \alpha_2, \ldots, \alpha_m\}$ with $\alpha_i = a_i^1 a_i^2 \ldots a_i^n \in \{0,1\}^n$, $1 \leq i \leq m$. We think of the assignments from $\Pi_{3\text{SAT}}$ as if mapping a variable set $\{x_1, x_2, \ldots, x_n\}$ to $\{0,1\}$.

The basis of the definition of $f(\Pi_{3\text{SAT}})$ for well-formed sets $\Pi_{3\text{SAT}}$ will be the set $S_{\Pi_{3\text{SAT}}}$ of triples that will be defined step by step during the proof.

First, we put the following $2n$ sets S_i' and $S_i'', 1 \leq i \leq n$, into $S_{\Pi_{3\text{SAT}}}$. Let $r = \binom{n-1}{2}$.

$$\begin{aligned} S_i' &= \{(s_i^j, x_i^j, t_i^{j+1}), 1 \leq j \leq r-1\} \cup \{(s_i^r, x_i^r, t_i^1)\} \\ S_i'' &= \{(s_i^j, \neg x_i^j, t_i^j), 1 \leq j \leq r\}. \end{aligned}$$

Since there will be no further triples in $S_{\Pi_{3\text{SAT}}}$ that contain any of the variables s_i^j and t_i^j we can establish the following claim.

Lemma 8.2.2. *Let* M *be a 3D-matching for* $S_{\Pi_{3\text{SAT}}}$. *Then for each* $i, 1 \leq i \leq n$, M *either completely contains* S_i' *or completely contains* S_i''. *If* M *contains* S_i' (S_i'') *then* $M \cap S_i'' = \varnothing$ ($M \cap S_i' = \varnothing$), *i.e., there are no triples from* S_i'' (S_i') *in* M.

Proof. Let M be a matching for $S_{\Pi_{3\text{SAT}}}$ and let i be an arbitrary natural number such that $1 \leq i \leq n$. Since s_i^1 has to be covered, one of the triples (s_i^1, x_i^1, t_i^2) and $(s_i^1, \neg x_i^1, t_i^1)$ is an element of M.

Let $(s_i^1, x_i^1, t_i^2) \in M$. So t_i^2 is already covered and it follows that s_i^2 can only be covered by (s_i^2, x_i^2, t_i^3). Iteratively, it follows that the triples (s_i^3, x_i^3, t_i^4), (s_i^4, x_i^4, t_i^5), $\ldots, (s_i^r, x_i^r, t_i^1)$ are members of M and hence $S_i' \subseteq M$ and $S_i'' \cap M = \varnothing$.

If we suppose that $(s_i^1, \neg x_i^1, t_i^1) \in M$ it follows that $S_i'' \subseteq M$ and $S_i' \cap M = \varnothing$ in an analogous manner. □

For the reduction, we need a correspondence between Boolean assignments and matchings of $S_{\Pi_{3SAT}}$. Lemma 8.2.2 provides the basis for this correspondence. We associate a matching M of S and the assignment α_M such that $\alpha_M(x_i) = 1$ if $S_i'' \subseteq M$ and $\alpha_M(x_i) = 1$ if $S_i'' \subseteq M$, $1 \leq i \leq n$. This way of correspondence seems to be non-canonical (S_i'' covers $\neg x_i^j, 1 \leq j \leq r$). But, in case of $S_i'' \subseteq M$ the positive elements $x_i^j, 1 \leq j \leq r$, still have to be covered, so the defined correspondence will be more canonical in the steps to come. Note that if $S_i'' \subseteq M$ ($S_i' \subseteq M$), there are $r = \binom{n-1}{2}$ versions of the variable x_i ($\neg x_i$), $1 \leq i \leq n$, that still must be covered.

Our intention is to define the set $S_{\Pi_{3SAT}}$ such that $S_{\Pi_{3SAT}}$ has exactly one 3D-matching for each assignment that is 3-compatible with Π_{3SAT}. Then we will map Π_{3SAT} to the set Π_{3DM} of associated 3D-matchings of $S_{\Pi_{3SAT}}$ and finally, we will see that $\Pi_{3SAT} \in$ Inverse-3SAT if and only if $\Pi_{3DM} \in$ Inverse-3DM, a $\leq_m^p$-reduction.

We just defined a correspondence between matchings and assignments. So, the remaining part of the construction will deal with the issue to "check" whether the assignment defined by a matching's usage of the sets S_i' respectively $S_i'', 1 \leq i \leq n$, is 3-compatible with Π_{3SAT}. Since 3-compatibility means $\{x_{a_k}, x_{b_k}, x_{c_k}\}$-compatibility for all possible triplets $\{x_{a_k}, x_{b_k}, x_{c_k}\}, 1 \leq k \leq \binom{n}{3}, 1 \leq a_k < b_k < c_k \leq n$, we can (and will) check 3-compatibility by checking $\{x_{a_k}, x_{b_k}, x_{c_k}\}$-compatibility, for all $k, 1 \leq k \leq \binom{n}{3}$. We will do so by inserting a couple of sets with upper index k, for each $k, 1 \leq k \leq \binom{n}{3}$, that will realize the test of $\{x_{a_k}, x_{b_k}, x_{c_k}\}$-compatibility.

Recall that $\Pi_{3SAT} = \{\alpha_1, \alpha_2, \ldots, \alpha_m\}$ with $\alpha_i = a_i^1 a_i^2 \ldots a_i^n \in \{0,1\}^n$, $1 \leq i \leq m$. Now, for each $k, 1 \leq k \leq \binom{n}{3}$, we define the auxiliary set

$$\Pi_{3SAT}^k = \{b^1 b^2 b^3 \in \{0,1\}^3 : (\exists i \in \{1, \ldots, m\})[b^1 = a_i^{a_k} \wedge b^2 = a_i^{b_k} \wedge b^3 = a_i^{c_k}]\}$$

of partial assignments from Π_{3SAT} (restricted to $\{x_{a_k}, x_{b_k}, x_{c_k}\}$). Let $\ell^k = |\Pi_{3SAT}^k|$ be the number of partial assignments in Π_{3SAT}^k (note that $\ell^k \leq 8$) and $\Pi_{3SAT}^k = \{\beta_1^k, \beta_2^k, \ldots, \beta_{\ell^k}^k\}$. For each element $\beta_i^k = b_i^{1,k} b_i^{2,k} b_i^{3,k}, 1 \leq i \leq \ell^k$, of Π_{3SAT}^k we add the following set S_i^k of three triples to $S_{\Pi_{3SAT}}$:

$$S_i^k = \{(u_1^k, *_1^k x_{a_k}, v_1^{i,k}), (u_2^k, *_2^k x_{b_k}, v_2^{i,k}), (u_3^k, *_3^k x_{c_k}, v_3^{i,k})\},$$

where

$$*_{i'}^k x_j = \begin{cases} x_j, & \text{if } b_i^{i',k} = 1 \\ \neg x_j, & \text{if } b_i^{i',k} = 0 \end{cases}$$

for $1 \leq i' \leq 3, j \in \{a_k, b_k, c_k\}$. Note that we did not specify the version (the upper index) of the participating literals $x_{a_k}, \neg x_{a_k}, x_{b_k}, \neg x_{b_k}, x_{c_k}$, and $\neg x_{c_k}$ so far. We want to use different versions of a literal for different (upper) indices k, hence we have to count up the version of a literal for each triplet $\{x_{a_k}, x_{b_k}, x_{c_k}\}$, the variable is contained in. So each appearance of a variable x_j in the sets $S_i^k, 1 \leq i \leq \ell^k$ gets the upper index ℓ if $\ell - 1 = |\{\{x_{a_s}, x_{b_s}, x_{c_s}\} : x_j \in \{x_{a_s}, x_{b_s}, x_{c_s}\} \wedge 1 \leq s < k\}|$. A negated variable gets the same upper index as the positive one. The fact that

each variable appears in $\binom{n-1}{2}$ of the $\binom{n}{3}$ triplets belatedly explains why we created exactly $\binom{n-1}{2}$ versions of each variable.

To illustrate the rather simple idea behind the complicated definition of the sets S_i^k, we give a small example.

Example 8.2.3. *Let* $\Pi_{3\text{SAT}} = \{(11001), (11010), (10111), (01110)\}$. *If we consider the first three variables* x_1, x_2, x_3 *(the triple with the upper index* $k = 1$*), we obtain the set* $\Pi_{3\text{SAT}}^1 = \{(110), (101), (011)\}$ *(with* $\ell^1 = 3$*) of partial assignments, restricted to* x_1, x_2, x_3. *Hence, the sets*

$$\begin{aligned} S_1^1 &= \{\{(u_1^1, x_1, v_1^{1,1}), (u_2^1, x_2, v_2^{1,1}), (u_3^1, \neg x_3, v_3^{1,1})\} \text{ (due to (110))} \\ S_2^1 &= \{\{(u_1^1, x_1, v_1^{2,1}), (u_2^1, \neg x_2, v_2^{2,1}), (u_3^1, x_3, v_3^{2,1})\} \text{ (due to (101))} \\ S_3^1 &= \{\{(u_1^1, \neg x_1, v_1^{3,1}), (u_2^1, x_2, v_2^{3,1}), (u_3^1, x_3, v_3^{3,1})\} \text{ (due to (011))} \end{aligned}$$

are put into $S_{\Pi_{3\text{SAT}}}$. *Recall Lemma 8.2.2 and the defined correspondence between assignments and matchings. Thus, for instance* $S_1^1 \subseteq M$ *implies that the assignment* α *that is associated to* M *equals* (110) *when restricted to* x_1, x_2 *and* x_3. *Since, forthcoming steps will make sure that exactly one of the sets* S_1^1, S_2^1, *and* S_3^1 *is included in each 3D-matching* M *of* $S_{\Pi_{3\text{SAT}}}$, *we will have that the associated assignments* α *is* $\{x_1, x_2, x_3\}$*-compatible with* $\Pi_{3\text{SAT}}$.

The following lemma is the generalization of this idea.

Lemma 8.2.4. *Let* M *be a 3D-matching for* $S_{\Pi_{3\text{SAT}}}$ *that includes the set* S_i^k, *for some* $1 \leq k \leq \binom{n}{3}$, *and some* $1 \leq i \leq \ell^k$, *and let* α_M *be the assignment associated with* M. *Then,* α_M *is* $\{x_{a_k}, x_{b_k}, x_{c_k}\}$*-compatible with* $\Pi_{3\text{SAT}}$.

Proof. Recall that all versions of x_i ($\neg x_i$) are already covered by S_i' (S_i'') if the assignment α_M associated with M assigns 0 (1) to x_i. Hence by the construction of S_i^k, we have that $S_i^k \subseteq M$ implies that $\beta_i'^k$, which is the restriction of an assignment from $\Pi_{3\text{SAT}}$ to $\{x_{a_k}, x_{b_k}, x_{c_k}\}$, agrees with α_M on x_{a_k}, x_{b_k}, and x_{c_k}. Thus α_M is $\{x_{a_k}, x_{b_k}, x_{c_k}\}$-compatible with $\Pi_{3\text{SAT}}$. □

Fix any $k, 1 \leq k \leq \binom{n}{3}$, for the moment. To benefit from Lemma 8.2.4, we want to make sure that each 3D-matching M completely includes one of the sets $S_i^k, 1 \leq i \leq \ell^k$. In order to achieve that, we put the following auxiliary sets T_i^k and $T_i'^k$ into $S_{\Pi_{3\text{SAT}}}$, for each partial assignment $\beta_i'^k$, $1 \leq i \leq \ell^k$, of $\Pi_{3\text{SAT}}^k$:

$$\begin{aligned} T_i^k &= \{(y_1^{i,k}, y_2^{i,k}, v_1^{i,k}), (y_2^{i,k}, y_3^{i,k}, v_2^{i,k}), (y_3^{i,k}, y_1^{i,k}, v_3^{i,k})\} \text{ and} \\ T_i'^k &= \{(y_1^{i,k}, y_1^{i,k}, z_1^k), (y_2^{i,k}, y_2^{i,k}, z_2^k), (y_3^{i,k}, y_3^{i,k}, z_3^k)\}. \end{aligned}$$

We do so, for each $k, 1 \leq k \leq \binom{n}{3}$. Since there will be no more triples added to $S_{\Pi_{3\text{SAT}}}$ that contain $u_1^k, u_2^k, u_3^k, z_1^k, z_2^k, z_3^k$ or any of the variables $y_1^{i,k}, y_2^{i,k}, y_3^{i,k}, v_1^{i,k}, v_2^{i,k}$, and $v_3^{i,k}, 1 \leq k \leq \binom{n}{3}, 1 \leq i \leq \ell^k$, we can establish the following Lemma.

Lemma 8.2.5. *Let M be a 3D-matching for $S_{\Pi_{3SAT}}$. Then for each k, $1 \leq k \leq \binom{n}{3}$, M includes the set S_i^k for some $i \in \{1, \ldots, \ell^k\}$ and $M \cap S_j^k = \varnothing$ for all $j, 1 \leq j \leq \ell^k, j \neq i$. Furthermore if S_i^k is contained in M, it follows that $T_i'^k \subseteq M$ and $T_j^k \subseteq M$, for all $j \neq i, 1 \leq j \leq \ell^k$.*

Proof. Note that u_1^k can only be covered by one of the triples $(u_1^k, *_1^k x_{a_k}, v_1^{i,k})$, $1 \leq i \leq \ell^k$. Let u_1^k be covered by $(u_1^k, *_1^k x_{a_k}, v_1^{i,k}) \in S_i^k$, for some $1 \leq i \leq \ell^k$. Hence, $y_1^{i,k}$ must be covered by $(y_1^{i,k}, y_1^{i,k}, z_1^k)$. Analogous to the proof of Lemma 8.2.2, it follows that $(y_3^{i,k}, y_3^{i,k}, z_3^k) \in M$ and $(y_2^{i,k}, y_2^{i,k}, z_2^k) \in M$, thus $T_i'^k \subseteq M$. So the elements $v_2^{i,k}$ and $v_3^{i,k}$ can only be covered by $(u_2^k, *_2^k x_{b_k}, v_2^{i,k})$ and $(u_3^k, *_3^k x_{c_k}, v_2^{i,k})$. It follows that $S_i^k \subseteq M$. Since S_i^k covers u_1^k, u_2^k and u_3^k, it follows that no triples from $S_j^k, j \neq i$ are in M. To cover the elements $v_1^{j,k}, v_2^{j,k}$, and $v_3^{j,k}$, $j \neq i, 1 \leq j \leq \ell^k$, M must contain the sets $T_j^k, j \neq i, 1 \leq j \leq \ell^k$. □

This concludes the construction of the set $S_{\Pi_{3SAT}}$ and we define $X_{S_{\Pi_{3SAT}}} := X(S_{\Pi_{3SAT}})$, $Y_{S_{\Pi_{3SAT}}} := Y(S_{\Pi_{3SAT}})$, and $Z_{S_{\Pi_{3SAT}}} := Z(S_{\Pi_{3SAT}})$.

Lemma 8.2.6. *Let Π_{3SAT} be a well-formed set of proofs for V_{3SAT} and let $S_{\Pi_{3SAT}}$ be the above constructed set. There exists a one-to-one correspondence between the 3D-matchings for $S_{\Pi_{3SAT}}$ and the assignments that are 3-compatible with Π_{3SAT}, namely the one, that maps a matching M to the assignment α_M associated with M.*

Proof. Let Π_{3SAT} be a well-formed set of proofs for V_{3SAT} and let $S_{\Pi_{3SAT}}$ be the set constructed from Π_{3SAT} as described above. First, we will show that if M is a 3D-matching for $S_{\Pi_{3SAT}}$ then the assignment α_M associated with M is 3-compatible with Π_{3SAT}.

So let M be a 3D-matching of $S_{\Pi_{3SAT}}$ and let α_M be the assignment associated with M. Let $k \in \mathbb{N}$ such that $1 \leq k \leq \binom{n}{3}$. Lemma 8.2.5 ensures that M includes S_i^k for an $i, 1 \leq i \leq \ell^k$, and hence by Lemma 8.2.4 it follows that α_M is $\{x_{a_k}, x_{b_k}, x_{c_k}\}$-compatible with Π_{3SAT}. So α_M is $\{x_{a_k}, x_{b_k}, x_{c_k}\}$-compatible with Π_{3SAT} for all $k, 1 \leq k \leq \binom{n}{3}$, that is, α_M is 3-compatible with Π_{3SAT}.

Now, it is sufficient to show that for each 3-compatible assignment α, there exists exactly one matching M_α for $S_{\Pi_{3SAT}}$ that corresponds to α. So let α be 3-compatible with Π_{3SAT}. For each $i, 1 \leq i \leq n$, the Boolean value that α assigns to x_i uniquely determines whether S_i' or S_i'' is included in a matching M_α, associated with α. So we start building up M_α by inserting the proper set S_i' respectively S_i'' for each $i, 1 \leq i \leq n$. This step is obviously unique.

Note that if $\alpha(x_i) = 1$ ($\alpha(x_i) = 0$), for a number $i, 1 \leq i \leq n$, then $S_i'' \subseteq M_\alpha$ ($S_i' \subseteq M_\alpha$) covers all versions $\neg x_i^j$ (x_i^j), $1 \leq j \leq \binom{n-1}{2}$, of $\neg x_i$ (x_i) and all versions x_i^j ($\neg x_i^j$), $1 \leq j \leq \binom{n-1}{2}$, of x_i ($\neg x_i$) still have to be covered.

We pick one number $k, 1 \leq k \leq \binom{n}{3}$, and hence one triplet $\{x_{a_k}, x_{b_k}, x_{c_k}\}$. By Lemma 8.2.5, it follows that each matching for $S_{\Pi_{3SAT}}$ includes one of the sets $S_i^k, 1 \leq i \leq \ell^k$. By the construction of the sets $S_i^k, 1 \leq i \leq \ell^k$, we have that exactly the set $S_{j'}^k$ must be put into M_α, that corresponds to the assignment $\beta_{j'}^k \in \Pi_{3SAT}^k$

that assigns the same values to x_{a_k}, x_{b_k}, and x_{c_k} as α does. Since α is 3-compatible and thus $\{x_{a_k}, x_{b_k}, x_{c_k}\}$-compatible with $\Pi_{3\text{SAT}}$ such an assignment $\beta^k_{j'}$ actually exists. So we have to put $S^k_{j'}$ into M_α for this unique index j'. Lemma 8.2.5 provides furthermore that $T'^k_{j'}$ and $T^k_j, j \neq j', 1 \leq j \leq \ell^k$, have to be put into M_α. If we do so for all $k, 1 \leq k \leq \binom{n}{3}$, a closer look shows that all elements from $X_{\Pi_{3\text{SAT}}}, Y_{\Pi_{3\text{SAT}}}$, and $Z_{\Pi_{3\text{SAT}}}$ are covered exactly once and hence M_α is a 3D-matching for $S_{\Pi_{3\text{SAT}}}$. Note that in each step, the triples put into M_α were uniquely determined. Thus, M_α is the only matching of $S_{\Pi_{3\text{SAT}}}$ that corresponds to α. □

Now, we can define $f(\Pi_{3\text{SAT}})$ to be the set $\Pi_{3\text{DM}} = \{M_\alpha : \alpha \in \Pi_{3\text{SAT}}\}$, where M_α is the matching constructed in the proof of Lemma 8.2.6. To conclude the proof of the theorem, we have to show that the obviously polynomial-time-computable function f realizes the needed reduction.

First, assume that $\Pi_{3\text{SAT}} \in$ Inverse-3SAT. Hence, $\Pi_{3\text{SAT}}$ is well-formed and closed under 3-compatibility. By Lemma 8.2.6, it follows that $S_{\Pi_{3\text{SAT}}}$ exactly has the proofs $f(\Pi_{3\text{SAT}}) = \{M_\alpha : \alpha \in \Pi_{3\text{SAT}}\}$. Hence, $f(\Pi_{3\text{SAT}}) \in$ Inverse-3DM.

Conversely, let $\Pi_{3\text{SAT}} \notin$ Inverse-3SAT. It follows that $\Pi_{3\text{SAT}}$ is not well-formed or $\Pi_{3\text{SAT}}$ is not closed under 3-compatibility. As defined in the beginning of the proof, if $\Pi_{3\text{SAT}}$ is not well-formed, f maps it to a fixed non-member of Inverse-3DM. So let $\Pi_{3\text{SAT}}$ be well-formed but not closed under 3-compatibility. Hence, there exists an assignment $\beta \notin \Pi_{3\text{SAT}}$ that is 3-compatible with $\Pi_{3\text{SAT}}$ and by Lemma 8.2.6, it follows that M_β is a 3D-matching for $S_{\Pi_{3\text{SAT}}}$. The fact that M_β is not in $f(\Pi_{3\text{SAT}})$ is obvious. So for the proof that $f(\Pi_{3\text{SAT}}) = \{M_\alpha : \alpha \in \Pi_{3\text{SAT}}\}$ is not in Inverse-3DM it suffices to show that M_β is a matching for the candidate $c_{3\text{DM}}(f(\Pi_{3\text{SAT}}))$ of the constructed proof set $f(\Pi_{3\text{SAT}})$.

Recall that the candidate $c_{3\text{DM}}(\Pi_{3\text{DM}})$ for a well-formed set $\Pi_{3\text{DM}} = \{M_1, M_2, \ldots, M_r\}$ is $(\bigcup_{i=1}^{r} M_i, X_{M_1}, Y_{M_1}, Z_{M_1})$. Since each $M_\alpha \in f(\Pi_{3\text{SAT}})$ covers exactly $X_{S_{\Pi_{3\text{SAT}}}}$, $Y_{S_{\Pi_{3\text{SAT}}}}$, and $Z_{S_{\Pi_{3\text{SAT}}}}$, it follows $c_{3\text{DM}}(f(\Pi_{3\text{SAT}})) = (\bigcup_{\alpha \in \Pi_{3\text{SAT}}} M_\alpha, X_{S_{\Pi_{3\text{SAT}}}}, Y_{S_{\Pi_{3\text{SAT}}}}, Z_{S_{\Pi_{3\text{SAT}}}})$. If M_β is completely included in $\bigcup_{\alpha \in \Pi_{3\text{SAT}}} M_\alpha$, it is easy to see that M_β is a 3D-matching for $c_{3\text{DM}}(f(\Pi_{3\text{SAT}}))$, which would imply the desired fact that $f(\Pi_{3\text{SAT}})$ is not Inverse-3DM.

In order to show that M_β is included in $\bigcup_{\alpha \in \Pi_{3\text{SAT}}} M_\alpha$, let $(\mathsf{a}, \mathsf{b}, \mathsf{c})$ be an arbitrary triple from M_β. Hence, $(\mathsf{a}, \mathsf{b}, \mathsf{c})$ is included in one of the sets $S'_i, S''_i, 1 \leq i \leq \binom{n-1}{2}$, or in one of the sets $S^k_i, T^k_i, T'^k_i, 1 \leq k \leq \binom{n}{3}$ and $1 \leq i \leq \ell^k$.

First, let $(\mathsf{a}, \mathsf{b}, \mathsf{c}) \in S'_i$ (S''_i), for some $i, 1 \leq i \leq \binom{n-1}{2}$. By Lemma 8.2.2, it follows that S'_i $(S''_i) \subseteq M_\beta$ and thus $\beta(x_i) = 0$ (1). Since β is 3-compatible with $\Pi_{3\text{SAT}}$ there exists an assignment $\alpha_j \in \Pi_{3\text{SAT}}$ such that $\alpha_j(x_i) = 0(1)$. It follows that S'_i $(S''_i) \subseteq M_{\alpha_j}$ and hence $(\mathsf{a}, \mathsf{b}, \mathsf{c}) \in \bigcup_{\alpha \in \Pi_{3\text{SAT}}} M_\alpha$.

Second, assume that $(\mathsf{a}, \mathsf{b}, \mathsf{c})$ is in S^k_i, T^k_i or $T'^k_i, 1 \leq k \leq \binom{n}{3}$ and $1 \leq i \leq \ell^k$. Since β is 3-compatible and in particular $\{x_{a_k}, x_{b_k}, x_{c_k}\}$-compatible with $\Pi_{3\text{SAT}}$, there exists an assignment $\alpha_j \in \Pi_{3\text{SAT}}$ such that β and α_j assign the same truth-values to the variables from $\{x_{a_k}, x_{b_k}, x_{c_k}\}$. By the construction of M_β from β it follows that M_β includes the same sets $S^k_i, 1 \leq i \leq \ell^k$, $T^k_i, 1 \leq i \leq \ell^k$, and $T'^k_i, 1 \leq$

$i \leq \ell^k$ as M_{α_j}. Hence, $(\mathsf{a}, \mathsf{b}, \mathsf{c}) \in M_{\alpha_j} \subseteq \bigcup_{\alpha \in \Pi_{3\mathrm{SAT}}} M_\alpha$. So we have that $M_\beta \subseteq \bigcup_{\alpha \in \Pi_{3\mathrm{SAT}}} M_\alpha$. It follows that $M_\beta \notin f(\Pi_{3\mathrm{SAT}})$ is a matching for $c_{3\mathrm{DM}}(f(\Pi_{3\mathrm{SAT}}))$ and thus $f(\Pi_{3\mathrm{SAT}}) \notin$ Inverse-3DM. □

Combining the above theorem with the results from [KS99, Che03, Krü05], we have the coNP-completeness of the inverse problems of all six basic NP-complete problems [GJ79].

Theorem 8.2.7 ([KS99, Che03, Krü05]). *The inverse problems* Inverse-3SAT, Inverse-3DM, Inverse-VC, Inverse-Clique, Inverse-HC, and Inverse-Partition are coNP-complete.

8.3 Inverse problems and universal verifiers

When dealing with the complexity of alternative solutions, different universal verifiers for one and the same problem can be treated as equivalent, because the corresponding alternative solution problems have the same computational complexity. It is rather natural to also discuss the complexity of inverse problems, defined with respect to different universal verifiers for one and the same NP-problem. One might conjecture that these inverse problems also have the same complexity. Although we can not disprove this conjecture, we give an example which strongly motivates the opposite conjecture.

8.3.1 Preliminaries

For the following section we will need the notion of one-way functions. So we give a very short introduction. For a more detailed introduction to one-way functions see, e.g., [Pap94].

There are different definitions for one-way functions. We will use the following one that can be found in e.g. [Pap94].

Definition 8.3.1. *A function $f : \Sigma^* \to \Sigma^*$ is called a* one-way function *if and only if the following conditions hold.*

1. *f is one-to-one, and for all $x \in \Sigma^*$, $|x|^{(1/k)} \leq |f(x)| \leq |x|^k$, for some $k > 0$. That is, $f(x)$ is at most polynomial longer or shorter than x.*

2. *f is computable in polynomial time, $f \in$ FP .*

3. *$f^{-1} \notin$ FP. That is, there is no polynomial-time algorithm, which, given y, either computes x such that $f(x) = y$, or returns "no", if no such x exists.*

Even under the assumption that P $\neq$ NP, there is no guarantee that such functions exists. However, the stronger precondition that P $\neq$ UP leads to the existence of one-way functions.

Theorem 8.3.2 ([Pap94]). P $\neq$ UP *if and only if one-way functions exist.*

Since, it is widely believed that P $\neq$ UP, we also believe in the existence of one-way functions. Some suspects for one-way functions are a one-to-one version of *integer multiplication*, *exponentiation modulo a prime* or *discrete logarithms*.

8.3.2 A plausible conjecture

Here, we give the argumentation for the conjecture that different universal verifiers for the same problem can induce inverse problem of different computational complexity.

We consider the problem SAT of satisfiable Boolean formulas in CNF. In Chapter 4 we have seen that

$$V_{\mathrm{SAT}} = \{(F, \alpha) : F \text{ is in CNF and } F(\alpha) = 1\}$$

is a universal verifier for SAT. Since for each set of assignments over a given variable set X there is a CNF-formula having exactly these satisfying assignments, it follows that each syntactically correct set of assignments is contained in Inverse-$(\mathrm{SAT}, V_{\mathrm{SAT}})$. So Inverse-$(\mathrm{SAT}, V_{\mathrm{SAT}})$ can be decided in polynomial-time.

Theorem 8.3.3. Inverse-$(\mathrm{SAT}, V_{\mathrm{SAT}})$ is in P.

Now, also consider the verifier

$$V'_{\mathrm{SAT}} = \{(F, (f(F), \alpha)) : F \text{ is in 3CNF and } F(\alpha) = 1\}$$

for SAT, where f is a one-way function. Note that verifying if $(F, (c, \alpha)) \in V_{\mathrm{SAT}}$ can be done in polynomial time because $f(F)$ can be computed and thus, $c = f(F)$ can be tested. So, V'_{SAT} is actually a verifier for SAT.

It is easy to see that V'_{SAT} is also universal for SAT.

Theorem 8.3.4. *The set V'_{SAT} is a universal verifier for* SAT.

Proof. Since V_{SAT} is universal it suffices to give a *gp*-reduction $(\mathrm{SAT}, V_{\mathrm{SAT}}) \leq^p_{gp}$ $(\mathrm{SAT}, V'_{\mathrm{SAT}})$. It can easily be verified that the functions $f = id$ and $g : (F, \alpha) \mapsto (f(F), \alpha)$ are polynomial-time computable and realize the reduction. □

So, assuming that different universal verifiers induce inverse problems of the same complexity, Inverse-$(\mathrm{SAT}, V'_{\mathrm{SAT}})$ should also be in P. Below, we will give strong arguments that this is not the case.

Let $\Pi = \{(c_1, \alpha_1), \ldots, (c_k, \alpha_k)\}$ be a set of proofs for V'_{SAT} with $c_i = c_j$, for $1 \leq i, j \leq k$. It is easy to see that $\Pi \in$ Inverse-$(\mathrm{SAT}, V'_{\mathrm{SAT}})$, if and only if $c_1 = c_2 = \cdots = c_k = f(F)$ for some CNF-formula F and $F(\alpha) = 1 \leftrightarrow \alpha \in \Pi$. Note, that depending on $\Pi' = \{\alpha_1, \ldots, \alpha_k\}$, there can be a huge amount of CNF-formulas F_l satisfying the latter. It not hard to see, that this amount can even be exponential in the size of Π.

Thus, a potential algorithm for Inverse-$(\mathrm{SAT}, V'_{\mathrm{SAT}})$ has to evaluate, if for one of these formulas holds $f(F_l) = c_1$. Assuming that no polynomial-time algorithm can extract properties of F from $c_1 = f(F)$, this can only be done by computing $f(F_l)$ for all these candidates F_l and checking if $f(F_l) = c_1$, which costs exponential time.

So, we do not expect that a polynomial-time algorithm for Inverse-$(\mathrm{SAT}, V'_{\mathrm{SAT}})$ exists, which leads to the opposite conjecture that the inverse problems of different universal verifiers for the same problem can have substantially different computational complexities. Obviously, the above argumentation depends on the existence of a one-way function f and the assumption that no polynomial-time algorithm can extract properties of x from $f(x)$. However, under these reasonable assumptions holds the following conjecture.

Conjecture 8.3.5. *The inverse problems defined with respect to two different universal verifiers for one and the same* NP*-problem may have a very different computational complexity.*

8.4 Conclusions

Similar to our proof of the coNP-completeness of Inverse-HC [KH06] we have shown the coNP-completeness of Inverse-3DM, that is, we have used the notion of 3-compatibility to reduce Inverse-3SAT to Inverse-3DM. This result completes the analysis of the inverse problem of the six basic NP-complete problems [GJ79]. They are all coNP-complete. However, there still are a lot of NP-problems to discuss. Probably one can also use the idea of 3-compatibility to reduce Inverse-3SAT to some more inverse problems.

Open Problem 8. *Study some more inverse problems!*

In the second part, we examined the notion of universal verifiers in the context of inverse problems. It turned out that two different universal verifiers for the same NP-problem highly likely may induce inverse problems of different complexity (Conjecture 8.3.5).

Open Problem 9. *Verify or disprove Conjecture 8.3.5!*

Bibliography

[AB92] M. Agrawal and S. Biswas. Universal relations. In *Proceedings of the Seventh Annual Structure in Complexity Theory Conference (Boston, MA, 1992)*, pages 207–220, Los Alamitos, CA, 1992. IEEE Comput. Soc. Press.

[AH98] A. M. Abdelbar and S. M. Hedetniemi. Approximating MAPs for belief networks is NP-hard and other theorems. *Artificial Intelligence*, 102(1):21–38, 1998.

[BDG88] J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural complexity. I*, volume 11 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Berlin, 1988.

[BDG90] J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural complexity. II*, volume 22 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Berlin, 1990.

[Ber66] R. Berger. The undecidability of the domino problem. *Memoirs American Mathematical Society*, 66:1–72, 1966.

[Ber05] T. Berg. Komplexität inverser Probleme (german). Master's thesis, Friedrich-Schiller-University Jena, April 2005.

[BKL+02] P. Berman, M. Karpinski, L. L. Larmore, W. Plandowski, and W. Rytter. On the complexity of pattern matching for highly compressed two-dimensional texts. *J. Comput. System Sci.*, 65(2):332–350, 2002.

[BP89] M. Bern and P. Plassmann. The Steiner problem with edge lengths 1 and 2. *Inform. Process. Lett.*, 32(4):171–176, 1989.

[BST99] C. Bazgan, M. Santha, and Z. Tuza. On the approximation of finding a(nother) Hamiltonian cycle in cubic Hamiltonian graphs. *Journal of Algorithms*, 31(1):249–268, 1999.

[Che03] H. Chen. Inverse NP problems. In *Mathematical foundations of computer science 2003*, volume 2747 of *Lecture Notes in Computer Science*, pages 338–347. Springer, Berlin, 2003.

[Chr76] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, 1976.

[Chv73] V. Chvátal. On the computational complexity of finding a kernel. Technical report, Université de Montréal, 1973. Report No. CRM-300, Centre de Recherches Mathématiques.

[CK97] P. Crescenzi and V. Kann. Approximation on the web: a compendium of NP optimization problems. In *Randomization and approximation techniques in computer science (Bologna, 1997)*, volume 1269 of *Lecture Notes in Comput. Sci.*, pages 111–118. Springer, Berlin, 1997.

[Coo71] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of ACM STOC'71*, pages 151–158, 1971.

[dB04] M. de Bondt. On the ASP-completeness of cryptarisms. Technical Report 0419, Department of Mathematics, Radboud University of Nijmegen, 2004.

[EJ77] S. Even and D. S. Johnson. mentioned in [GJ79] as unpublished results, 1977.

[FHT97] S. Fischer, L. Hemaspaandra, and L. Torenvliet. Witness-isomorphic reductions and local search. In *Complexity, logic, and recursion theory*, volume 187 of *Lecture Notes in Pure and Appl. Math.*, pages 207–223. Dekker, New York, 1997.

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

[GJP77] M. R. Garey, D. S. Johnson, and C. H. Papadimitriou. mentioned in [GJ79] as unpublished results, 1977.

[GJT76] M. R. Garey, D. S. Johnson, and R. E. Tarjan. The planar hamiltonian circuit problem is NP-complete. *SIAM Journal on Computing*, 5(4), 1976.

[GW95] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. Assoc. Comput. Mach.*, 42(6):1115–1145, 1995.

[Hal93] M. M. Halldórsson. Approximating the minimum maximal independence number. *Inform. Process. Lett.*, 46(4):169–172, 1993.

[HMRS98] H. B. Hunt, III, M. V. Marathe, V. Radhakrishnan, and R. E. Stearns. The complexity of planar counting problems. *SIAM J. Comput.*, 27(4):1142–1167 (electronic), 1998.

[HQ85] H. W. Hamacher and M. Queyranne. K best solutions to combinatorial optimization problems. *Ann. Oper. Res.*, 4(1-4):123–143, 1985.

[Kan94] V. Kann. Polynomially bounded minimization problems that are hard to approximate. *Nordic J. Comput.*, 1(3):317–331, 1994. Selected papers of the 20th International Colloquium on Automata, Languages and Programming (ICALP 93) (Lund, 1993).

[Kar72] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations (Proc. Sympos., IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y., 1972)*, pages 85–103. Plenum, New York, 1972.

[KH06] M. Krüger and H. Hempel. Inverse Hamiltonian Cycle and inverse 3Dimensional Matching are coNP-complete. In *ISAAC*, pages 243–252, 2006.

[KL95] Ker-I Ko and Chih-Long Lin. On the complexity of min-max optimization problems and their approximation. In *Minimax and applications*, volume 4 of *Nonconvex Optim. Appl.*, pages 219–239. Kluwer Acad. Publ., Dordrecht, 1995.

[Kra99] A. Krawczyk. The complexity of finding a second Hamiltonian cycle in cubic graphs. *J. Comput. System Sci.*, 58(3):641–647, 1999.

[Krü05] M. Krüger. Weitere Hamiltonkreise in Graphen und inverse Hamiltonkreisprobleme (german). Master's thesis, Friedrich-Schiller-University Jena, July 2005.

[KS99] D. Kavvadias and M. Sideri. The inverse satisfiability problem. *SIAM J. Comput.*, 28(1):152–163 (electronic), 1999.

[LL78] N. Lynch and R. Lipton. On structure preserving reductions. *SIAM J. Comput.*, 7(2):119–126, 1978.

[McP03] B. P. McPhail. The complexity of puzzles: NP-completeness results for nurikabe and minesweeper. bachelor thesis, The Division of Mathematics and Natural Sciences, Reed College, Portland, 2003.

[OM90] P. Orponen and H. Manila. On approximation preserving reductions: Complete problems and robust measures, 1990.

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

[PS76] C. H. Papadimitriou and K. Steiglitz. Some complexity results for the traveling salesman problem. In *Eighth Annual ACM Symposium on Theory of Computing (Hershey, Pa., 1976)*, pages 1–9. Assoc. Comput. Mach., New York, 1976.

[PS82] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Prentice-Hall Inc., Englewood Cliffs, N.J., 1982.

[PY91] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *J. Comput. System Sci.*, 43(3):425–440, 1991.

[PY93] C. H. Papadimitriou and M. Yannakakis. The traveling salesman problem with distances one and two. *Math. Oper. Res.*, 18(1):1–11, 1993.

[Rog67] H. Rogers. *Theory of recursive functions and effective computability*. McGraw-Hill Book Co., New York, 1967.

[RZ00] G. Robins and A. Zelikovsky. Improved Steiner tree approximation in graphs. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (San Francisco, CA, 2000)*, pages 770–779, New York, 2000. ACM.

[Set02] T. Seta. The complexity of puzzles, cross sum and their another solution problems (ASP). Senior Thesis, Department of Infomation Science, the Faculty of Science, the University of Tokyo, 2002.

[SM73] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time: preliminary report. In *Fifth Annual ACM Symposium on Theory of Computing (Austin, Tex., 1973)*, pages 1–9. Assoc. Comput. Mach., New York, 1973.

[SM93] Y. Saruwatari and T. Matsui. A note on K-best solutions to the Chinese postman problem. *SIAM J. Optim.*, 3(4):726–733, 1993.

[Sto76] L. J. Stockmeyer. The polynomial-time hierarchy. *Theoret. Comput. Sci.*, 3(1):1–22 (1977), 1976.

[Tho78] A. G. Thomason. Hamiltonian cycles and uniquely edge colourable graphs. *Ann. Discrete Math.*, 3:259–268, 1978. Advances in graph theory (Cambridge Combinatorial Conf., Trinity College, Cambridge, 1977).

[Tut46] W. T. Tutte. On Hamiltonian circuits. *J. London Math. Soc.*, 21:98–101, 1946.

[Uma99] C. Umans. Hardness of approximating Σ_2^p minimization problems. In *40th Annual Symposium on Foundations of Computer Science (New York, 1999)*, pages 465–474. IEEE Computer Soc., Los Alamitos, CA, 1999.

[Uma01] C. Umans. The minimum equivalent DNF problem and shortest implicants. *J. Comput. System Sci.*, 63(4):597–611, 2001. Special issue on FOCS 98 (Palo Alto, CA).

[UN96] N. Ueda and T. Nagao. NP-completeness results for nonogram via parsimonious reductions. Technical report, Tokyo Institute of Technology, 1996.

[vdPLSvdV99] E. S. van der Poort, M. Libura, G. Sierksma, and J. A. A. van der Veen. Solving the k-best traveling salesman problem. *Comput. Oper. Res.*, 26(4):409–425, 1999. The traveling salesman problem.

[vEB97] P. van Emde Boas. The convenience of tilings. In *Complexity, logic, and recursion theory*, volume 187 of *Lecture Notes in Pure and Appl. Math.*, pages 331–363. Dekker, New York, 1997.

[WC96] Zheng Wang and J. Crowcroft. Quality-of-service routing for supporting multimedia applications. *IEEE Journal of Selected Areas in Communications*, 14(7):1228–1234, 1996.

[Wes96] D. B. West. *Introduction to graph theory*. Prentice Hall Inc., Upper Saddle River, NJ, 1996.

[Wra76] C. Wrathall. Complete sets and the polynomial-time hierarchy. *Theor. Comput. Sci.*, 3(1):23–33, 1976.

[WW86] K. Wagner and G. Wechsung. *Computational complexity*, volume 21 of *Mathematics and its Applications (East European Series)*. D. Reidel Publishing Co., Dordrecht, 1986.

[YG80] M. Yannakakis and F. Gavril. Edge dominating sets in graphs. *SIAM J. Appl. Math.*, 38(3):364–372, 1980.

[YS02] T. Yato and T. Seta. Complexity and completeness of finding another solution and its application to puzzles. In *In Proceedings of the National Meeting of the Information Processing Society of Japan (IPSJ)*, 2002.

Ehrenwörtliche Erklärung

Hiermit erkläre ich,

- daß ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe
- daß ich die Hilfe eines Promotionsberaters nicht in Anspruch genommen habe und daß Dritte weder unmittelbar noch mittelbar geldwerte Leistungen von mir für Arbeiten erhalten haben, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen
- daß ich die Dissertation noch nicht als Prüfungsarbeit für eine staatliche oder andere Wissenschaftliche Prüfung eingereicht habe.

Jena, den 21.01.2008

Michael Krüger

Lebenslauf

Persönliche Daten

Name	Michael Krüger
Geburt	09. 01. 1980 in Jena

Schulausbildung

1986–1990	Polytechnische Oberschule, Jena
1990–1994	Albert Schweitzer Gymnasium, Jena
1994–1998	Carl Zeiss Gymnasium mit naturwissenschaftlichen Spezialklassen, Jena
06/1998	Abitur

Studium

10/1999–07/2005	Studium der Mathematik an der Friedrich-Schiller-Universität (FSU) Jena
07/2005	Abschluss des Studiums mit dem Grad Diplom-Mathematiker Thema der Diplomarbeit: Weitere Hamiltonkreise in Graphen und inverse Hamiltonkreisprobleme

Akademische Laufbahn

seit 10/2005	Doktorant am Institut für Informatik der FSU gefördert durch ein Graduiertenstipendium des Landes Thüringen