Multiobjective Optimization and Language Equations

Dissertation zur Erlangung des
naturwissenschaftlichen Doktorgrades
der Julius-Maximilians-Universität Würzburg

vorgelegt von

Christian Reitwießner

Würzburg, 2011.

Eingereicht am: 21.12.2011
bei der Fakultät für Mathematik und Informatik
1. Gutachter: PD Dr. Christian Glaßer
2. Gutachter: Prof. Dr. Pierre McKenzie
3. Gutachter: Prof. Dr. Victor Selivanov
Tag der mündlichen Prüfung: 26.3.2012

I would like to express my gratitude to all those who directly or indirectly contributed to this thesis.

Special thanks go to my advisor Christian Glaßer. This work would not exist without his continuing support. In particular, I want to thank him for introducing me into research at an early stage.

I am happy that Klaus Wagner gave me the opportunity to work at his chair and I would like to thank him for readily integrating me in any discussion.

My gratitude also goes to Alexander Okhotin for sharing his insight and for hosting me in Turku twice.

It was a pleasure for me to work together or have inspiring discussions with Daniel Meister, Pierre McKenzie, Victor Selivanov, Heinz Schmitz and Edith and Lane Hemaspaandra.

I am glad I had Elmar Böhler, Stephen Travers and Maximilian Witek as my colleagues, who rapidly turned into friends.

Finally, I want to thank my parents, my family and of course Julia.

Abstract

Practical optimization problems often comprise several incomparable and conflicting objectives. When booking a trip using several means of transport, for instance, it should be fast and at the same time not too expensive. The first part of this thesis is concerned with the algorithmic solvability of such multiobjective optimization problems. Several solution notions are discussed and compared with respect to their difficulty. Interestingly, these solution notions are always equally difficulty for a single-objective problem and they differ considerably already for two objectives (unless P = NP). In this context, the difference between search and decision problems is also investigated in general. Furthermore, new and improved approximation algorithms for several variants of the traveling salesperson problem are presented. Using tools from discrepancy theory, a general technique is developed that helps to avoid an obstacle that is often hindering in multiobjective approximation: The problem of combining two solutions such that the new solution is balanced in all objectives and also mostly retains the structure of the original solutions.

The second part of this thesis is dedicated to several aspects of systems of equations for (formal) languages. Firstly, conjunctive and Boolean grammars are studied, which are extensions of context-free grammars by explicit intersection and complementation operations, respectively. Among other results, it is shown that one can considerably restrict the union operation on conjunctive grammars without changing the generated language. Secondly, certain circuits are investigated whose gates do not compute Boolean values but sets of natural numbers. For these circuits, the equivalence problem is studied, i. e. the problem of deciding whether two given circuits compute the same set or not. It is shown that, depending on the allowed types of gates, this problem is complete for several different complexity classes and can thus be seen as a (parametrized) representative for all those classes.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Computational tasks are usually characterized by specifying a set of valid inputs and, for each of the inputs, a set of valid outputs. Examples of such specifications are the following:

1. Input: A set of integers $\{x_1, x_2, \ldots, x_r\}$.
   Output: "yes" if there is a subset of $\{x_1, \ldots, x_r\}$ that sums to $\frac{1}{2}(x_1 + x_2 + \cdots + x_r)$, "no" otherwise.
2. Input: A system of linear equations.
   Output: A solution to the system or "no" if no solution exists.
3. Input: A finite set of places on a map.
   Output: A shortest round-trip that visits all of the places.
4. Input: Distances and toll costs for a road map, two points on the map.
   Output: A route between the two points such that no other route is both shorter and cheaper.
5. Input: A finite set of relatives where some are at odds with each other.
   Output: "yes" if they can be placed at a round table such that nobody is at odds with a neighbor, "no" otherwise.
6. Input: Two arithmetical expressions built from natural numbers by applying the operations $+$, $\cdot$ and $()^2$, e. g. $(3+4)^2 \cdot (7+2)$ and $(3 \cdot (5+2))^2$.
   Output: "yes" if the two expressions describe the same number, "no" otherwise.
7. Input: A text and a syntax specification given by a context-free grammar.
   Output: "yes" if the text conforms to the specification, "no" otherwise.

Such a task is considered to be solvable by computers if one can provide an *algorithm* for it, i. e. a list of precise instructions how to get from any possible input x to one valid output for x. In complexity theory, one is interested in the amount of computational resources (like, for instance, computing time, memory space or randomness) that are needed when solving computational tasks. Since the size of the inputs can vary and an algorithm generally needs more resources on larger inputs, one considers the resources needed as a function of the input size. If for a task there is some k and an algorithm that takes at most n^k steps for inputs of size n bits, this task is said to be solvable in

polynomial time. If a task (or problem) is solvable in polynomial time, it is considered to be feasible in practice. From the list above, only the problems 2, 4 and 7 are known to be solvable in polynomial time.

Many computational tasks ask for an object possessing a certain property specified by the input. If for a given object, it is easy to check if it has the property, then there is a general strategy to solve these tasks: Simply try all possible objects and check them one after another. For problem 1, for example, we can try all subsets of $\{x_1, \ldots, x_r\}$, compute their sum and compare it to $\frac{1}{2}(x_1 + \cdots + x_r)$. This method is called solving by *brute force* or *exhaustive search* and it of course fails if the number of candidates is infinite and we additionally do not know if any of the objects has the property at all. For many problems, though, the number of candidates can be bounded. Furthermore, it is often the case that each candidate can be described by at most n^k bits for some k, where n is again the size of the input. If additionally the check can be carried out in polynomial time, we arrive at a problem in the class NP (NP stands for *nondeterministic polynomial time*, a term that derives from another, equivalent way to define this class). Formally, only the problem that asks whether such an object exists or not (i. e. a simple yes/no-question) lies in the class NP. This distinction into *search problems* (actually output the object) and *decision problems* (only determine if such an object exists or not) will be explored in detail later. The problems 1 and 5 above are typical problems in NP. For problem 1, each possible candidate, i. e. each subset of $\{x_1, \ldots, x_r\}$ can be described by a list of r bits, where the ith bit specifies if x_i is included in the set or not.

The most important open question in complexity theory is whether all problems in NP can be solved in polynomial time or not, the so-called P-NP-problem. Intuitively, simply checking if a given object has a certain property should be easier than actually finding such an object, but until now, it is not clear how to prove this in general. Furthermore, there is meta-mathematical evidence suggesting that the P-NP-problem cannot be solved with current techniques. Problems 1 and 5 actually belong to the hardest problems in NP, the so-called NP-*complete problems*. It can be shown that one of them is solvable in polynomial time if and only if all of them are solvable in polynomial time. Many open questions in complexity theory are connected to the P-NP-problem in the sense that an answer would also solve the P-NP-problem. Hence, it is very unlikely that these open questions can be solved directly. Instead, the relations between different assumptions are usually investigated.

Optimization problems form a large class of problems that arise in real-world settings. The task is to find a solution that maximizes or minimizes a given objective function. An example of such a problem is problem 3 from the list above. Here, the objective function is the length of the round-trip and the goal is to minimize this length. Typically, optimization problems are formulated as search problems. However, one can always associate a decision problem to each search problem. For problem 3, we can add a natural number x to the input and ask if there is a solution to problem 3 that is better than x. For many important optimization problems, the associated decision problem is NP-complete, which is also the case here.

Apart from the differences between search and decision problems already mentioned, this thesis will be concerned with optimization problems, and more specifically, optimization problems with more than one objective. Furthermore, we will investigate problems resembling problem 6 and 7 and see that arithmetical expressions and syntax specifications

using context-free grammars are actually quite similar, which is not apparent at first sight.

Multiobjective Optimization

As already noted, problem 3 from the list above is an optimization problem where the tours are the solutions and their length is to be minimized. The problem of shortest paths is another minimization problem where the shortest route between two points in a road network has to be found given a table of distances between any two junctions on a road. This problem is often not useful in reality, since the distance (or even the time) is not the only quantity that is to be optimized in practical route planning. Other quantities are perhaps the probability of traffic jams, the quality of the road or the toll costs. The optimization problem can of course be easily extended to include more than one objective function and one then arrives at a problem similar to problem 4. The difficulties arise when possible routes are compared: Since there can be routes that are faster and other routes that are cheaper, it is not clear anymore what the best route is. The user of such a planning system may have certain preferences about the solution based on additional information or simply experience, but this information is not available to an algorithm that solves the formal problem. One is often tempted to combine several objective functions into a single one (yielding a quantity like money), for example using a weighted sum, but without further information about the problem, it is not evident how to apply the weights, i. e. the price of an hour of time is not clear. The only option is to consider all objectives to be equally important and try to find all so-called *Pareto-optimal* solutions. A solution is Pareto-optimal if there is no other solution that is at least as good in all objectives and better in at least one objective.

Unfortunately, computing all Pareto-optimal solutions is often not feasible in practice. The reasons are that there are often simply too many Pareto-optimal solutions and furthermore, they are hard to compute. There are several other ways to at least give the user an impression of this set and sometimes also the ability to navigate through it to obtain a specific solution. The first part of this thesis will be concerned with a comparison of these so-called *solution notions* and furthermore the presentation of several algorithms that solve multiobjective extensions of problem 3, the so-called traveling salesperson problem.

Language Equations

The second part of this thesis is concerned with the complexity of certain types of expressions or equations. From the above list, the problems 6 and 7 fall into this category. Problem 7 is called *parsing* and is quite well understood and feasible in practice. It is an essential step when compiling source code to an executable program.

A context-free grammar more or less consists of a set of rules that specify how one symbol can be replaced by a string of symbols. If one now applies these rules beginning from a start symbol, one obtains more and more complicated strings. Consider the rules $S \to aSa$, $S \to bSb$ and $S \to c$, i. e. every S can be either replaced by aSa, by bSb or by c. When we start from S and apply the first rule, we obtain aSa. If we then use the second rule, we get $abSba$. A further application of the first and then the third rule results in $abacaba$. Here, none of the rules can be applied anymore. Symbols that can be replaced

are called *nonterminals* (only S in the example), the other symbols are called *terminals* (a, b and c here). Since we can choose from multiple rules in each step, several so-called words can be generated by this grammar. If we disregard all words that still contain the nonterminal S, we obtain the (formal) *language* generated by the grammar. Here, this language is the set of all words over the alphabet $\{a, b, c\}$ that equal their reversed word and have a c in the center. So if a syntax specification is given by a context-free grammar, then the parsing problem is the question whether a given word can be generated by the grammar.

Since each nonterminal symbol can have multiple rules, there is an implicit logical disjunction in the semantics of context-free grammars. Okhotin [Okh01, Okh04] extended the context-free grammars by an explicit conjunction and negation in the rules and thus obtained the so-called *conjunctive* and *Boolean grammars*, respectively. Here, the right-hand sides of rules contain explicit intersection and complementation operators. They can thus be regarded as expressions that use concatenation, intersection and complementation (for Boolean grammars) over terminal and nonterminal symbols. These grammars are more powerful than context-free grammars as they can generate languages that cannot be generated by any context-free grammar. Nevertheless, the parsing problem is efficiently solvable also for conjunctive and Boolean grammars.

If only one terminal symbol is allowed, the conjunctive and Boolean grammars turn out to be still surprisingly powerful. It is well-known that context-free grammars with only one terminal symbol can only generate *regular* languages. Regular languages can be defined as the class of languages where the task of checking if a given word is in the language can be solved by an algorithm whose memory requirement is constant, independent of the size of the input. In the second part of this thesis, we will investigate Boolean grammars in general and also study the parsing problem for Boolean grammars with only one terminal symbol.

Observe that if the alphabet consists of only one symbol, then any word in a language is completely determined by its length. This means that a language can be characterized by a subset of the natural numbers. Furthermore, two words are concatenated by adding their lengths and this can be generalized to concatenating whole languages. So there is a natural correspondence between expressions over a single-letter alphabet containing the operations of concatenation, union, intersection and complementation and the respective expressions over sets of natural numbers.

These expressions are thus similar to the expressions considered in problem 6. Stockmeyer and Meyer [SM73] studied the problem of testing whether two expressions over sets of natural numbers characterize the same set, the so-called *equivalence problem*. They obtained that the complexity of this problem heavily depends on the operations allowed. We will study similar problems concerning expressions where repeated subexpressions are encoded more succinctly.

We now explain the specific results obtained in the individual chapters in more detail.

Chapter Two

This chapter contains all relevant definitions in complexity theory and the theory of formal languages. It is not meant as an introduction to the theory of formal languages

or complexity theory but rather as a reference for the notions and notations used in this thesis.

Chapter Three

We investigate the principal differences between search and decision problems in this chapter: A decision problem is solved by answering a yes/no-question, whereas a search problem is more complicated and generalizes this concept: Here, the task is to construct an object possessing a certain property defined by the input or stating that it does not exist. In contrast to decision problems, which are forced to exactly one answer for each input, search problems have some kind of freedom: If there are several objects that possess the property, the algorithm can choose between them. On the other hand, when such algorithms are used as subroutines, we cannot pose any restriction on the object we get. These two concepts are used to compare several classes of search and decision problems with respect to *polynomial-time Turing-equivalence*. Roughly, two problems are polynomial-time Turing-equivalent, if one problem can be solved in polynomial time when it has access to a (hypothetical) routine that solves the other problem in polynomial time and vice-versa. The main result in this chapter is that (under some assumption) there are search problems that are inequivalent to any decision problem. The above mentioned freedom for search problems is the main reason why this result is true. A conclusion of this chapter is that the usual complexity of decision problems is not the right measure for the complexity of search problems.

Chapter Four

In this chapter we give a complexity-theoretical foundation for multiobjective optimization problems. Multiobjective optimization has been studied for quite some time, but there has not been a thorough complexity-theoretic treatment of these problems so far. In contrast to single-objective problems, it is initially not clear what it means to solve such a problem. We formalize multiobjective optimization problems and define several solution notions used in the literature. It turns out that most of these notions coincide if there is only one objective, which is the reason why they are not differentiated in single-objective optimization. Furthermore, we give several examples of multiobjective optimization problems in this framework.

Chapter Five

Next, we compare the solution notions defined in chapter four with regard to their complexity. Here, the preliminary work of the chapter on search and decision problems will turn out to be helpful.

As a main result we obtain that for any three problems in NP, there is a single multiobjective optimization problem where one solution notion is equivalent to the first problem, another to the second and yet another one is equivalent to the third problem. This means that the complexities for multiobjective optimization problems can differ drastically depending on the solution notion considered. Hence it is important to distinguish between

these notions to avoid simply speaking of *the* complexity of a multiobjective optimization problem.

Furthermore, we show that some solution notions are inherently search problems, while some are inherently decision problems. So the conclusion drawn in the preceding chapter is also applicable here: The complexity of some solution notions of multiobjective optimization problems cannot be described in terms of decision problems (under some complexity-theoretical assumption).

Chapter Six

Many important single-objective optimization problems are NP-complete and this is even more the case for multiobjective optimization problems. As already mentioned, it is widely believed that NP-complete problems cannot be solved in polynomial time. Hence, one turns to so-called approximate solutions: It is often not crucial to obtain the best possible solution as long as one gets a solution together with the guarantee that this solution is within a certain factor of being optimal. Like in the setting where one is interested in exactly optimal solutions, the situation is more complicated for multiobjective optimization than it is in the single-objective case. We extend the solution notions from the previous chapter to approximate solution notions and again investigate the relations between them. For many notions, we can show how to transfer approximability results from one notion to the other. Furthermore, for some problems, approximations for single-objective variants directly transfer to the multiobjective case and thus existing algorithms can be reused.

Chapter Seven

Having laid the foundations for approximating multiobjective problems, we turn to specific problems. The traveling salesperson problem (problem 3 in the list above) is one of the most important optimization problems in complexity theory. We investigate several variants of the multiobjective traveling salesperson problem and improve the currently best known approximations. We use one of the variants to exemplify how to use discrepancy theory to design approximation algorithms for multiobjective problems: An idea that appears in many single-objective approximation algorithms is that if a set of objects that are assigned weights is divided into two parts, one of the two parts has at least half of the total weight. As it is, this idea is not usable in multiobjective optimization, where the weights are vectors, as one of the two parts can be heavier in the first and the other heavier in the second component. Using results from discrepancy theory, a given set of objects with assigned multidimensional weights can be divided into two parts of roughly the same weight in each component. This means that one of the two parts must have at least one half of the weight in each component minus some error. Thus the idea from single-objective optimization can be transferred to multiobjective optimization if a small error is acceptable.

Chapter Eight

We conclude the first part by improving the result from discrepancy theory used in the previous chapter in a way: We show that one can always assume that the objects are

divided into two parts in a special way. Suppose we want to assign the objects to the two parts sequentially according to a pre-specified order. This means we have to assign some objects to the first part, then some two the second part, the again some to the first part and so on. We show that there is a strategy such that the number of changes between the parts is bounded by a number that is independent of the number of objects. The result is shown by proving some kind of multidimensional intermediate value theorem. This observation could help in finding approximation algorithms for multiobjective problems that require the assignment to have this more convenient structure.

Chapter Nine

The first chapter of the second part is concerned with conjunctive and Boolean grammars. As already mentioned, conjunctive and Boolean grammars are extensions of context-free grammars by explicit set-theoretic intersection and complementation, respectively. We show that by extending the context-free grammars by an explicit intersection operation, multiple rules for one nonterminal (which represent some kind of union) are almost not needed anymore: Conjunctive grammars where for each nonterminal there is at most one rule that references other nonterminals generate the same languages as general conjunctive grammars. This restriction imposed on context-free grammars has already been studied by Greibach, Shi and Simonson [GSS92], who found out that in this case, a weaker class of languages is obtained.

In the second part of this chapter, we investigate the complexity of deciding whether a given word over a single-letter alphabet is generated by a given Boolean grammar or not, i. e. the complexity of the parsing or membership problem. Valiant [Val75] showed that the parsing problem for context-free grammars can be reduced to Boolean matrix multiplication. We use a similar technique and reduce the parsing problem for Boolean grammars over a single-letter alphabet first to online Boolean convolution and this problem in turn to integer multiplication.

Chapter Ten

If we disallow circular references to nonterminals in Boolean grammars over a single-letter alphabet, we arrive (by the already mentioned correspondence) at expressions or circuits over sets of natural numbers. For each subset of allowed operations (union, intersection, complementation, addition, multiplication), the complexity of the equivalence problem of these circuits is explored. We show that these problems are complete for a vast range of complexity classes from NL and P over Π_2^{P} up to PSPACE. Furthermore, we obtain similar results for the satisfiability problem for such circuits: In this problem, we introduce variables and ask if for a certain number there is an assignment to the variables such that the desired number is produced.

Publications

This thesis is based on the following publications in journals and refereed conference proceedings together with the results from the technical reports [GRW09] and [OR11]. Chapter 8 contains unpublished joint work with C. Glaßer and M. Witek.

[GHR+10] C. Glaßer, K. Herr, C. Reitwießner, S. D. Travers, and M. Waldherr. Equivalence problems for circuits over sets of natural numbers. *Theory of Computing Systems*, 46(1):80–103, 2010.

[GRSW10] C. Glaßer, C. Reitwießner, H. Schmitz, and M. Witek. Approximability and hardness in multi-objective optimization. In *Proceedings Computability in Europe (CiE)*, volume 6158 of *Lecture Notes in Computer Science*. Springer Verlag, 2010.

[GRTW10] C. Glaßer, C. Reitwießner, S. D. Travers, and M. Waldherr. Satisfiability of algebraic circuits over sets of natural numbers. *Discrete Applied Mathematics*, 158(13):1394–1403, 2010.

[OR10] A. Okhotin and C. Reitwießner. Conjunctive grammars with restricted disjunction. *Theoretical Computer Science*, 411(26-28):2559–2571, 2010.

[GRW11] C. Glaßer, C. Reitwießner, and M. Witek. Applications of discrepancy theory in multiobjective approximation. In *Proceedings Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 13 of *LIPIcs*, pages 55–65. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011.

[FGL+12] F. Lipp, C. Reitwießner, K. Fleszar, C. Glaßer and M. Witek. Structural complexity of multiobjective NP search problems. In *Proceedings 10th Latin American Theoretical Informatics Symposium (LATIN)*, 2012, to appear.

Chapter 2

Basics, Notations and Terminology

2.1 Sets, Numbers, Vectors and Functions

Sets and Numbers. The *empty set* is denoted by $\emptyset$. For a set A, let $2^A = \{X \mid X \subseteq A\}$ denote its *power set* and $\#A$ its *cardinality* if it is finite. Let $\mathbb{N} = \{0, 1, 2, \dots\}$ be the set of *natural numbers* (including zero), $\mathbb{Z}$ the *integers* and $\mathbb{N}^+$ the *positive integers*. The *rationals* and the *reals* are denoted by $\mathbb{Q}$ and $\mathbb{R}$, respectively. Finally, $\mathbb{B} = \{\text{false}, \text{true}\} = \{0, 1\}$ is the *Boolean semiring*. The *difference* of two sets A and B is denoted as $A - B = \{x \in A \mid x \notin B\}$. The *complement* of a set A with respect to a base set B (usually $\mathbb{N}$) is written as $\overline{A}$. Operators on objects are extended to sets of these objects by applying the operator to all combinations. For example for the addition, we get $X + Y = \{x + y \mid x \in X, y \in Y\}$.

Intervals (of reals) are written as $[a, b]$ or $[a, b)$, where a square bracket denotes a closed end and a rounded bracket an open end. The *signum* of a number $x \in \mathbb{R}$ is written as $\operatorname{sgn}(x)$ and its *absolute value* by $|x|$. Though more often, the latter denotes the *length* of the encoding of an object defined below. Finally, $\lfloor x \rfloor$ and $\lceil x \rceil$ denote the largest integer that is not larger than x and the smallest integer that is not smaller than x, respectively.

As a convention, variables normally represent natural numbers and Greek variables are often reals or rationals. Elements of Cartesian products are written as tuples (row-vectors) or, if the vector character wants to be stressed as vectors (column-vectors).

Vector Spaces. We usually assume a standard basis for all vector spaces and access the components of a vector $x \in F^k$ as $x_1, \dots, x_k$. Sometimes, we use different finite sets as index sets for the components of vectors, for example $\{0, 1, \dots, k-1\}$ or just some unspecified finite set Y, while we stress this by specifying the vector space as F^Y. For $x = (x_1, \dots, x_k)^{\mathrm{T}}$, we write $x[a..b]$ for the subvector $(x_a, x_{a+1}, \dots, x_b)^{\mathrm{T}}$ of x. Let 1_n be the all-ones-vector of dimension n and $\operatorname{diag}(x_1, \dots, x_n)$ be the $n \times n$ diagonal matrix with diagonal $x_1, \dots, x_n$. Finally, we define the usual preorder relations on an n-dimensional vector space (over an ordered field) such that $x \leq y$ holds if and only if $x_i \leq y_i$ for all $1 \leq i \leq n$.

On $\mathbb{R}^k$, the *p-norms* are defined as $\|(x_1, \dots, x_k)^T\|_p = (\sum_{i=1}^k |x_i|^p)^{1/p}$ for real numbers $p \geq 1$ and the *maximum norm* is $\|(x_1, \dots, x_k)^T\|_\infty = \max_i |x_i|$. The determinant of a square matrix A is written as $\det(A)$.

Functions. A function $f\colon A \to B$ is often extended to subsets C of A by $f(C) = \{f(x) \mid x \in C\}$. For $D \subseteq B$ we then also write $f^{-1}(D) = \{x \in A \mid f(x) \in D\}$.

Let log denote the logarithm to base two and $\log^*$ the iterated logarithm defined as

$$\log^*(x) = \min\{n \in \mathbb{N} \mid \underbrace{(\log \circ \log \circ \cdots \circ \log)}_{n}(x) \leq 1\}.$$

For $n \in \mathbb{N}$, $\mathrm{bin}(n)$ denotes the *binary representation* of n and $|n| = |\mathrm{bin}(n)|$ is the length of the binary representation of a number (see below for a formal definition of the length of a word). We likewise apply $|\cdot|$ to other objects and assume that they are appropriately encoded in binary. Note that $|\cdot|$ can also stand for the absolute value of a number, though this meaning is less often used.

Definition 2.1. For $f\colon \mathbb{N} \to \mathbb{N}$ define the sets $O(f)$, $o(f)$ and $\Theta(f)$ as follows.

$$\begin{aligned}
O(f) &= \{g\colon \mathbb{N} \to \mathbb{N} \mid \exists k \in \mathbb{N}\colon g(n) \leq k \cdot f(n) \\
&\qquad \text{for all but finitely many } n\} \\
\Theta(f) &= \{g\colon \mathbb{N} \to \mathbb{N} \mid \exists k_1, k_2 \in \mathbb{N}\colon \tfrac{1}{k_1} \cdot f(n) \leq g(n) \leq k_2 \cdot f(n) \\
&\qquad \text{for all but finitely many } n\} \\
o(f) &= \{g\colon \mathbb{N} \to \mathbb{N} \mid \forall k \in \mathbb{N}\colon g(n) \leq \tfrac{1}{k} \cdot f(n) \\
&\qquad \text{for all but finitely many } n\}
\end{aligned}$$

We will use these *Landau-symbols* also in an informal way, e. g. in $4n^2 + 2n = 4\,O(n^2) = O(n^2) \leq O(2^n)$, where we assume appropriately quantified representatives.

2.2 Formal Languages

Alphabets, Words and Languages. Throughout this thesis, the symbol Σ represents a finite set, also called *alphabet*. A *word (over Σ)* is a finite sequence of symbols (from Σ). The *empty word* is denoted by ε. The *length of a word w* is denoted by $|w|$. The *concatenation* of two words u and v is written as $u \cdot v$ or simply as uv. A *(formal) language* is a set of words and we extend the concatenation to languages by $K \cdot L = KL = \{uv \mid u \in K, v \in L\}$. Let $L^0 = \{\varepsilon\}$ and $L^{i+1} = L^i \cdot L$ for all $i \in \mathbb{N}$ and define the *iteration* of L as $L^* = \bigcup_{i \in \mathbb{N}} L^i$. For an alphabet Σ, the set of all words over Σ then equals Σ^* and for any $n \in \mathbb{N}$ the set of words of length exactly n is written as Σ^n and the set of words of length at most n is defined as $\Sigma^{\leq n} = \bigcup_{0 \leq i \leq n} \Sigma^i$. As we will need these special languages later, we define the set of *words of odd length* over an alphabet Σ as $\textsc{Odd} = \bigcup_{i \in \mathbb{N}} \Sigma^{2i+1}$ and the set of *words of even length* as $\textsc{Even} = \bigcup_{i \in \mathbb{N}} \Sigma^{2i}$.

Operations on Languages. For a language L over an implied alphabet Σ, we define the *complement* of L with respect to Σ^*, i. e. $\overline{L} = \Sigma^* - L$.

The *quotient* of a language with a singleton is defined as follows: For all $L \subseteq \Sigma^*$ and $u \in \Sigma^*$, the languages $u^{-1}L = \{w \mid uw \in L\}$ and $Lu^{-1} := \{w \mid wu \in L\}$ are the *left* and *right quotients* of L with u, respectively. This operation is extended to languages as $K^{-1}L = \{v \mid \exists u \in K\colon uv \in L\}$ and $LK^{-1} = \{u \mid \exists v \in K\colon uv \in L\}$ for $K, L \subseteq \Sigma^*$.

Regular Expressions and Context-Free Grammars. A *regular expression* over the alphabet Σ is a finite expression over $\emptyset$ and $\{a\}$ for any $a \in \Sigma$ that uses the operations of concatenation, union and iteration. A language is *regular* if it can be represented by a regular expression.

A *context-free grammar* is a tuple $G = (\Sigma, N, S, R)$, where Σ is a finite alphabet (the set of *terminal symbols*), N is a finite set (of *nonterminal symbols*), $S \in N$ is the *start symbol* and $R \subseteq N \times (\Sigma \cup N)^*$ is the set of *rules*, where a rule $(A, w) \in R$ is written as $A \to w$. A word $uvw \in (\Sigma \cup N)^*$ can be *derived from* $uAw \in (\Sigma \cup N)^*$ *in a single step (via* G*)*, $uAw \overset{G}{\Longrightarrow} uvw$, if there is a rule $A \to v \in R$. If we have $w_0 \overset{G}{\Longrightarrow} w_1 \overset{G}{\Longrightarrow} \ldots \overset{G}{\Longrightarrow} w_k$ for $k \in \mathbb{N}$ and $w_i \in (\Sigma \cup N)^*$, then w_k *can be derived from* w_0 *in exactly* k *steps (via* G*)*. Furthermore, the relation of *derivability (in zero or more steps via* G*)* for $u, v \in (\Sigma \cup N)^*$ is defined as $u \overset{G}{\Longrightarrow}^* v$ if and only if $u \overset{G}{\Longrightarrow}^k v$ for some $k \in \mathbb{N}$. For $u \in (\Sigma \cup N)^*$, the *language generated by* u is $L_G(u) = \{w \in \Sigma^* \mid u \overset{G}{\Longrightarrow}^* w\}$ and the *language generated by* G is $L(G) = L_G(S)$. A language that can be generated by a context-free grammar is called *context-free*.

2.3 Graph Theory

Graphs. A *directed edge (from* u *to* v*)* is a pair $e = (u, v)$ and an *undirected edge* is a set $e = \{u, v\}$, where $u \neq v$. In both cases, the edge e *connects* or is *incident to* the two *vertices* (or *nodes*) u and v.

A *directed (or undirected) (simple) graph* is a pair $G = (V, E)$, where V is a finite set (the *vertices* or *nodes of* G) and E is a finite set (the *edges of* G) of directed (or undirected) edges connecting vertices from V.

A *directed (or undirected) multigraph* is a pair $G = (V, E)$, where V is a finite set (the *vertices* or *nodes of* G) and E is a finite set containing elements (e, i) where e is a directed (or undirected) edge connecting vertices from V and $i \in \mathbb{N}$. In abuse of notation, (e, i) and e are sometimes identified. The set of vertices incident to the edge (e, i) is denoted by $[(e, i)]$. The edges of a multigraph are also called *multi-edges*.

If nothing else is specified, we assume that graphs and multigraphs are undirected.

For an arbitrary set X, an *X-(edge-)labeled (multi)graph* is a triple $G = (V, E, l)$, where (V, E) is a (multi)graph and $l \colon E \to X$. Attributes of G like being directed or undirected are inherited from (V, E). If addition is defined on X (usually, we have $X = \mathbb{N}^k$), then l is extended to sets of edges by summation. Similarly, an *X-vertex-labeled (multi)graph* is a triple $G = (V, E, l)$, where (V, E) is a (multi)graph and $l \colon V \to X$.

Degree. The *degree* of a vertex v in an undirected (multi)graph G is the number of edges that are incident to it, denoted by $\deg_G(v)$. In a directed (multi)graph G, an edge (u, v) (or $((u, v), i)$ for multigraphs) is an *incoming edge* for v and an *outgoing edge* for u. The number of incoming edges for a vertex is called its *indegree* and the number of outgoing edges its *outdegree*.

Walks, Paths and Cycles. In a directed or undirected (multi)graph $G = (V, E)$, a *walk (from* v_0 *to* v_m*)* is an alternating sequence $w = v_0, e_1, v_1, \ldots, e_m, v_m$, $m \geq 1$ of vertices and edges where e_i connects v_{i-1} and v_i (and e_i is an edge from v_{i-1} to v_i if it

is directed). The vertices v_0 and v_m are called endpoints of the walk. For simplicity, a walk will sometimes be identified by the set of its edges. The walk w is called *closed* if $v_0 = v_m$. It *visits* the vertices in its sequence and if a vertex v occurs exactly k times in $v_0, \ldots, v_{m-1}, v_m$ (in $v_0, \ldots, v_{m-1}$ if $v_0 = v_m$), the walk *visits v exactly k times*. The walk is called *spanning* if it visits all vertices from V and a *path* if no vertex is visited more than once. A closed path is called a *cycle*. Spanning paths (or cycles) are also called *Hamiltonian paths (or cycles)*. The walk $v_{i-1}, e_i, \ldots, e_j, v_j$ is a *(contiguous) subwalk* of w.

Properties of Graphs. A directed or undirected (multi)graph $G' = (V', E')$ is a *subgraph* of the directed or undirected (multi)graph $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. A directed or undirected graph is *complete* if it contains all possible edges. A directed or undirected (multi)graph is *strongly connected* if for each pair of vertices u, v there is a walk from u to v. It is *connected* if for each pair of vertices u, v there is a walk from u to v or from v to u and it is called *acyclic* if it does not contain any cycle. An acyclic connected undirected simple graph is called a *tree*. A subgraph $G' = (V', E')$ of an undirected (multi)graph $G = (V, E)$ is called a *spanning tree* if G' is a tree and $V' = V$. Often, G' is identified with E'.

Matchings and Cycle Covers. For a directed or undirected (multi)graph $G = (V, E)$ and a set of vertices $V' \subseteq V$, we call a set of edges $M \subseteq E$ a *matching of V' in G* if for each $v \in V'$ there is exactly one edge in M that is incident to v. A matching of $V' = V$ is called a *perfect matching*. In the same setting, a set of paths P in G is called a *path matching of V' in G* if each $v \in V'$ is the endpoint of exactly one path in P and no path is a cycle. A set of cycles C in a directed or undirected (multi)graph is called a *cycle cover* if each vertex is visited by exactly one cycle in C.

2.4 Complexity Theory

2.4.1 Turing Machines

The informal notion of an algorithm is formalized by the concept of a *Turing machine*. Exact definitions can be found in any textbook about computational complexity [Pap94, AB09]. Roughly, a Turing machine consists of a control with a finite number of states and a finite instruction table, furthermore, a finite number of tapes that are unbounded in both directions and divided into cells and a read/write head for each tape. The instruction table describes how the machine acts in each step depending on the current state and the symbols read from the tapes. Actions include state changes, movements of the heads to adjacent cells and writing single symbols to the tapes. The machine has special accepting and rejecting states and it stops execution if one of these states is reached. The machine is called *deterministic* if in each situation, exactly one instruction is applicable and *nondeterministic* if multiple instructions can be used. A *computation path* describes a possible sequence of successive situations and instructions for a nondeterministic machine. If the machine is deterministic, it can have a special write-only output tape and is called *Turing transducer* in this case.

We identify a Turing transducer M with the function $M\colon \mathbb{N} \to \mathbb{N}$ it computes defined in the following way: For all $x \in \mathbb{N}$, we run the machine where initially, the binary representation of x is written on the first tape. If the machine eventually stops in an accepting state and the output tape contains the binary representation of a number $y \in \mathbb{N}$, then $M(x) = y$. If it never stops, stops in a rejecting state or stops while the output tape does not contain the binary representation of a number (for example the special symbol $\perp$), then $M(x)$ is undefined.

Note that we allow functions to be *partial* as above, meaning that the actual codomain of a function $f\colon \mathbb{N} \to \mathbb{N}$ is not necessarily the whole set $\mathbb{N}$. Functions that are not partial are called *total*. This is useful in computability theory since a Turing machine does not need to halt on all inputs and it is difficult to determine the inputs where it does halt. Furthermore, if Turing machines work with objects that are not natural numbers as input or output, we assume a suitable encoding as natural numbers. We sometimes denote this encoding explicitly as $\langle\cdot\rangle$ and require that it is bijective in this case. In the section about encodings on page 31, we will explicitly define this encoding for tuples of natural numbers.

A function that is computed by a Turing machine is called *computable* and *recursive* if it is additionally total.

For a Turing machine M that is not a transducer, let $L(M)$ be the *set accepted* (or *set decided*) by M defined in the following way: For any $x \in \mathbb{N}$, we again run the machine as above. If the machine stops in an accepting state, then $x \in L(M)$ (and we say that M *accepts* the input x). If it does not stop at all or stops in a rejecting state, then $x \notin L(M)$ (M *rejects* the input x). If the machine halts on all inputs, then we say that M *decides* the set $L(M)$, otherwise it *accepts* the set. For nondeterministic machines, we also use the term *accept* even if the machine halts on all inputs.

A Turing transducer decides a set A by computing the *characteristic function* of A defined as

$$\mathrm{c}_A\colon \mathbb{N} \to \mathbb{N}, x \mapsto \begin{cases} 1, & \text{if } x \in A \\ 0, & \text{otherwise.} \end{cases}$$

Depending on the context, we identify a set and its characteristic function. Operations that can be carried out by Turing machines that halt on all inputs are called *effective* or *computable*.

2.4.2 Complexity Measures

We define several classes of sets (so-called *complexity classes*) by bounding the resources that are available to a Turing machine that decides the set. Formally, a complexity class is an arbitrary subset of $2^{\mathbb{N}}$ with no relation to computability or complexity. The elements of complexity classes are also called *languages* or *problems*. We already noted in the introduction that the resource bound of a Machine is a function of the input size. Here, the *size of a natural number* x, in symbols $|x|$ is the length of this number in binary representation. If objects other than natural numbers are used as input for a machine, where an implicit encoding is assumed, we also use $|\cdot|$ to denote the length of this encoding.

For a function $r\colon \mathbb{N} \to \mathbb{N}$, a (deterministic or nondeterministic) Turing machine M *runs in time* r if for every input $x \in \mathbb{N}$, the machine halts after at most $r(|x|)$ steps (on every computation path), with at most finitely many exceptions. The machine M *uses the*

space r if for every input $x \in \mathbb{N}$, it uses at most $r(|x|)$ tape cells (on every computation path), with at most finitely many exceptions, where read-only or write-only tapes (like the input tape or the oracle input tape defined later) are not taken into account. In these situations, we say that the machine M decides or accepts the set $L(M)$ (computes the function M) *in time* r or *in space* r.

Definition 2.2. Let $r\colon \mathbb{N} \to \mathbb{N}$ be some monotonically increasing function.

$$\begin{aligned}
\mathrm{DTIME}(r) &= \{A \subseteq \mathbb{N} \mid \text{there is a deterministic Turing machine } M \\
&\qquad \text{that decides } A \text{ in time } r\} \\
\mathrm{NTIME}(r) &= \{A \subseteq \mathbb{N} \mid \text{there is a nondeterministic Turing machine } M \\
&\qquad \text{that accepts } A \text{ in time } r\} \\
\mathrm{FDTIME}(r) &= \{f\colon \mathbb{N} \to \mathbb{N} \mid \text{there is a deterministic Turing machine } M \\
&\qquad \text{that computes } f \text{ in time } r\} \\
\mathrm{DSPACE}(r) &= \{A \subseteq \mathbb{N} \mid \text{there is a deterministic Turing machine } M \\
&\qquad \text{that decides } A \text{ in space } r\} \\
\mathrm{NSPACE}(r) &= \{A \subseteq \mathbb{N} \mid \text{there is a nondeterministic Turing machine } M \\
&\qquad \text{that accepts } A \text{ in space } r\} \\
\mathrm{FDSPACE}(r) &= \{f\colon \mathbb{N} \to \mathbb{N} \mid \text{there is a deterministic Turing machine } M \\
&\qquad \text{that computes } f \text{ in space } r\}
\end{aligned}$$

Using these generic classes, we can define several important classes considered in complexity theory.

Definition 2.3.

$$\begin{aligned}
\mathrm{L} &= \mathrm{DSPACE}(\log) & \mathrm{NL} &= \mathrm{NSPACE}(\log) \\
\mathrm{P} &= \bigcup_{k\in\mathbb{N}} \mathrm{DTIME}(n \mapsto n^k) & \mathrm{NP} &= \bigcup_{k\in\mathbb{N}} \mathrm{NTIME}(n \mapsto n^k) \\
\mathrm{FL} &= \mathrm{FDSPACE}(\log) & \mathrm{FP} &= \bigcup_{k\in\mathbb{N}} \mathrm{FDTIME}(n \mapsto n^k) \\
\mathrm{PSPACE} &= \bigcup_{k\in\mathbb{N}} \mathrm{DSPACE}(n \mapsto n^k) \\
\mathrm{E} &= \bigcup_{k\in\mathbb{N}} \mathrm{DTIME}(n \mapsto 2^{k\,n}) & \mathrm{NE} &= \bigcup_{k\in\mathbb{N}} \mathrm{NTIME}(n \mapsto 2^{k\,n}) \\
\mathrm{EXP} &= \bigcup_{k\in\mathbb{N}} \mathrm{DTIME}(n \mapsto 2^{n^k}) & \mathrm{NEXP} &= \bigcup_{k\in\mathbb{N}} \mathrm{NTIME}(n \mapsto 2^{n^k})
\end{aligned}$$

The class P (and FP for functions) is usually considered to contain the problems that are efficiently solvable. Consequently, we call a task that can be solved in polynomial time *efficient*, although this notion is not used too strictly.

There are some super-exponential time classes that play a special role in computational complexity. Using a padding argument, one can show that $\mathrm{EXP} \neq \mathrm{NEXP} \Rightarrow \mathrm{P} \neq \mathrm{NP}$. So if the assumption that $\mathrm{P} \neq \mathrm{NP}$ is not strong enough to show an assertion, one can assume the inequality of larger classes. We will use such assumptions to show that, for instance, there are search problems that are inequivalent to any decision problem.

Definition 2.4.

$$\mathrm{EE} = \bigcup_{k\in\mathbb{N}} \mathrm{DTIME}(n \mapsto 2^{2^{k\,n}}) \qquad \mathrm{NEE} = \bigcup_{k\in\mathbb{N}} \mathrm{NTIME}(n \mapsto 2^{2^{k\,n}})$$
$$\mathrm{EEE} = \bigcup_{k\in\mathbb{N}} \mathrm{DTIME}(n \mapsto 2^{2^{2^{k\,n}}}) \qquad \mathrm{NEEE} = \bigcup_{k\in\mathbb{N}} \mathrm{NTIME}(n \mapsto 2^{2^{2^{k\,n}}})$$

Note that some authors define these classes differently, for example the class EE is sometimes defined as $\bigcup_{k\in\mathbb{N}} \mathrm{DTIME}(n \mapsto 2^{k\,2^n})$.

2.4.3 Encoding for Tuples

At least for tuples of natural numbers we now define an encoding function to be used by Turing machines. This is done in two steps.

Let all tuples of natural numbers first be implicitly encoded in the following way, such that Turing machines can compute functions $\mathbb{N}^k \to \mathbb{N}$ and $\mathbb{N}^k \to \mathbb{N}$: For any $a \in \mathbb{N}$ let $\tilde{a} = a_1 a_1 a_2 a_2 \dots a_n a_n$ where $a_1 a_2 \dots a_n = \mathrm{bin}(a)$. A tuple $(x_1, x_2, \dots, x_k) \in \mathbb{N}^k$ is encoded as the number $\mathrm{bin}^{-1}(\tilde{x_1}01\tilde{x_2}01\dots01\tilde{x_k})$.

We replace this preliminary encoding by a family of bijective encodings $\mathrm{code}_k \colon \mathbb{N}^k \to \mathbb{N}$, $k \geq 1$ that satisfy the following conditions, where the complexity (and computability) of functions is defined using the preliminary encoding for input and output:

- code_k is a bijection that is monotone in each argument.
- code_k is computable and invertible in logarithmic space.
- $\bigcup_{k\geq 1} \mathbb{N}^k \to \mathbb{N}, (x_1, \dots, x_k) \mapsto \mathrm{code}_2(k, \mathrm{code}_k(x_1, \dots, x_k))$ is computable and invertible in polynomial time.

For the rest of this thesis, we will always denote $\mathrm{code}_k(x_1, \dots, x_k)$ as $\langle x_1, \dots, x_k \rangle$ and make sure that the number of arguments is either constant or additionally encoded. The *Cantor pairing function* π can be used to obtain such an encoding:

$$\pi(x_1, x_2) = \frac{1}{2}(x_1 + x_2)(x_1 + x_2 + 1) + x_2$$
$$\pi(x_1, x_2, \dots, x_k) = \pi(x_1, \pi(x_2, \dots, \pi(x_{k-1}, x_k)\dots))$$

2.4.4 Operators

Some complexity classes are not defined directly via machines but via operators that modify or combine other classes. Common operators are the *complement operator* co (not to be confused with the complementation operator $^-$) and the *projection*.

Definition 2.5. For a complexity class $\mathcal{C}$, the class of its complements is defined as

$$\mathrm{co}\mathcal{C} = \{\mathbb{N} - A \mid A \in \mathcal{C}\}.$$

Definition 2.6. For a set $A \subseteq \mathbb{N}$ and a total function $f\colon \mathbb{N} \to \mathbb{N}$, the *projection* of A with respect to f is defined as

$$\exists_f \cdot A = \{x \in \mathbb{N} \mid \exists y \in \mathbb{N}\colon y < 2^{f(|x|)} \wedge \langle x, y\rangle \in A\}$$

and generalized to all polynomial functions and to complexity classes $\mathcal{C}$ as

$$\exists^{\mathrm{P}} \cdot A = \{\exists_f \cdot A \mid f \text{ is a polynomial}\}$$
$$\exists^{\mathrm{P}} \cdot \mathcal{C} = \bigcup_{X \in \mathcal{C}} \exists^{\mathrm{P}} \cdot X$$

Using the projection, we obtain a different characterization of the important class NP as $\mathrm{NP} = \exists^{\mathrm{P}} \cdot \mathrm{P}$. Actually the projection operator is just another way to look at nondeterminism, though one has to be careful about the bound used in the projection. For example, it holds that $\mathrm{NL} = \exists^{\mathrm{P}} \cdot \mathrm{L}$, but $\exists^{\mathrm{P}} \cdot \mathrm{EXP} = \mathrm{EXP}$. Using this formalism, we can talk about *witnesses* and *verification*: For a set $A \in \exists^{\mathrm{P}} \cdot \mathrm{P}$, we can test if $x \in A$ by searching for a polynomially-sized witness y that can be verified in polynomial time.

If some class is not closed under complementation (or more likely, if this is unknown), one can investigate the *Boolean hierarchy* or more precisely, the *difference hierarchy* over that class:

Definition 2.7. Let $\mathcal{C}$ be a complexity class and $k \geq 1$.

- $\mathcal{C}(1) = \mathcal{C}$
- $\mathcal{C}(k+1) = \{A - B \mid A \in \mathcal{C}, B \in \mathcal{C}(k)\}$

The class $\mathcal{C}(k)$ is the *kth level of the Boolean hierarchy over* $\mathcal{C}$.

The following definition models the property of a set being not even close to lying in a certain complexity class.

Definition 2.8 ([BS85])**.** For a complexity class $\mathcal{C}$, an infinite and co-infinite set $L \subseteq \mathbb{N}$ is called $\mathcal{C}$*-bi-immune* if neither L nor $\overline{L}$ has an infinite subset in $\mathcal{C}$.

2.4.5 The Number of Witnesses and Probabilistic Classes

As we have seen in the introduction of the operator $\exists^{\mathrm{P}}\cdot$, an accepting computation path of a nondeterministic machine and a witness capture the same concept. By counting the number of witnesses, one can define several classes, among them classes that formalize the computational resource of *randomness*. The idea behind these randomized classes is that a potential witness (or a computation path) is chosen uniformly at random and the ratio of actual witnesses (or accepting paths) is the probability of acceptance.

Definition 2.9. For some (witness) set $X \subseteq \mathbb{N}$ and a function $f\colon \mathbb{N} \to \mathbb{N}$, let

$$\#_{X,f}\colon \mathbb{N} \to \mathbb{N},\ x \mapsto \#\{y \in \mathbb{N} \mid y < 2^{f(|x|)}, \langle x, y\rangle \in X\}$$

Using this notation, we can define several classes and again give an equivalent definition of the class NP. Let Pol be the set of all polynomials.

Definition 2.10.

$$\begin{aligned}
\mathrm{NP} &= \{A \subseteq \mathbb{N} \mid \exists X \in \mathrm{P}, p \in \mathrm{Pol}\, \forall x \in \mathbb{N}: \\
&\qquad (x \in A \iff \#_{X,p}(x) > 0)\} \\
\mathrm{UP} &= \{A \subseteq \mathbb{N} \mid \exists X \in \mathrm{P}, p \in \mathrm{Pol}\, \forall x \in \mathbb{N}: \\
&\qquad (x \in A \iff \#_{X,p}(x) > 0) \wedge \#_{X,p}(x) \leq 1\} \\
\mathrm{PP} &= \{A \subseteq \mathbb{N} \mid \exists X \in \mathrm{P}, p \in \mathrm{Pol}\, \forall x \in \mathbb{N}: \\
&\qquad (x \in A \iff \#_{X,p}(x) > \tfrac{1}{2}2^{p(|x|)})\} \\
\mathrm{RP} &= \{A \subseteq \mathbb{N} \mid \exists X \in \mathrm{P}, p \in \mathrm{Pol}\, \forall x \in \mathbb{N}: \\
&\qquad (x \in A \Rightarrow \#_{X,p}(x) > \tfrac{1}{2}2^{p(|x|)}) \wedge \\
&\qquad (x \notin A \Rightarrow \#_{X,p}(x) = 0)\} \\
\mathrm{BPP} &= \{A \subseteq \mathbb{N} \mid \exists X \in \mathrm{P}, p \in \mathrm{Pol}\, \forall x \in \mathbb{N}: \\
&\qquad (x \in A \Rightarrow \#_{X,p}(x) > \tfrac{2}{3}2^{p(|x|)}) \wedge \\
&\qquad (x \notin A \Rightarrow \#_{X,p}(x) < \tfrac{1}{3}2^{p(|x|)})\} \\
\mathrm{C_{=}P} &= \{A \subseteq \mathbb{N} \mid \exists X \in \mathrm{P}, p \in \mathrm{Pol}\, \forall x \in \mathbb{N}: \\
&\qquad (x \in A \iff \#_{X,p}(x) = \tfrac{1}{2}2^{p(|x|)})\} \\
\mathrm{C_{=}L} &= \{A \subseteq \mathbb{N} \mid \exists X \in \mathrm{L}, p \in \mathrm{Pol}\, \forall x \in \mathbb{N}: \\
&\qquad (x \in A \iff \#_{X,p}(x) = \tfrac{1}{2}2^{p(|x|)})\}
\end{aligned}$$

For the probabilistic classes PP, RP, and BPP, the ratio $\frac{\#_{X,p}(x)}{2^{p(|x|)}}$ is called the *probability of acceptance*.

We also define one class that is not so common but is a straightforward generalization of UP to super-exponential time and is needed later:

Definition 2.11.

$$\begin{aligned}
\mathrm{UEEE} = \{A \subseteq \mathbb{N} \mid \exists X \in \mathrm{P}, k \in \mathbb{N}\, \forall x \in \mathbb{N}: \\
(x \in A \iff \#_{X,f}(x) > 0) \wedge \#_{X,f}(x) \leq 1) \\
\text{where } f\colon n \mapsto 2^{2^{2^{kn}}}\}
\end{aligned}$$

2.4.6 Super-Exponential Time Classes and Sparse Sets

Next, we use standard padding techniques to construct several very sparse sets in NP under the assumption that certain super-exponential time classes do not coincide. These sets will be used in later chapters.

Proposition 2.12. Let $X = \{2^{2^x} \mid x \in \mathbb{N}\}$.

1. If $\mathrm{EE} \neq \mathrm{NEE}$ then there is some $B \subseteq X$ such that $B \in \mathrm{NP} - \mathrm{P}$.
2. If $\mathrm{EE} \neq \mathrm{NEE} \cap \mathrm{coNEE}$ then there is some $B \subseteq X$ such that $B \in (\mathrm{NP} \cap \mathrm{coNP}) - \mathrm{P}$.
3. If $\mathrm{NEE} \neq \mathrm{coNEE}$ then there is some $B \subseteq X$ such that $B \in \mathrm{NP} - \mathrm{coNP}$.
4. If $\mathrm{UEEE} \cap \mathrm{coUEEE} \neq \mathrm{NEEE} \cap \mathrm{coNEEE}$, then there exists a $B \in (\mathrm{NP} \cap \mathrm{coNP}) - (\mathrm{UP} \cap \mathrm{coUP})$ such that $B \subseteq \{t(i) + k \mid i \in \mathbb{N}, 0 \leq k < 2^i\}$ for $t(n) = 2^{2^{2^{2^n}}}$.

Proof. For $L \subseteq \mathbb{N}$ and $c \geq 1$ let $B(L,c) = \{2^{2^{x^c}} \mid x \in L\}$. Note that $B \subseteq X$. We claim:

$$L \in \mathrm{DTIME}(2^{2^{c \cdot n}}) \iff B(L,c) \in \mathrm{P} \tag{2.1}$$

$$L \in \mathrm{NTIME}(2^{2^{c \cdot n}}) \iff B(L,c) \in \mathrm{NP} \tag{2.2}$$

$$L \in \mathrm{coNTIME}(2^{2^{c \cdot n}}) \iff B(L,c) \in \mathrm{coNP} \tag{2.3}$$

If $L \in \mathrm{DTIME}(2^{2^{c \cdot n}})$ via the machine M, then $B(L,c) \in \mathrm{P}$ by the algorithm that on input $y = 2^{2^{x^c}}$ simulates $M(x)$. This simulation needs time $2^{2^{c \cdot |x|}} \leq 2^{2^{c \cdot (1+\log x)}} = 2^{2^c \cdot x^c} = (\log y)^{2^c} \leq |y|^{2^c}$, which is polynomial in $|y|$.

If on the other hand $B(L,c) \in \mathrm{P}$, then $L \in \mathrm{DTIME}(2^{2^{c \cdot n}})$ by the algorithm that on input x simulates the deterministic polynomial-time algorithm for $B(L,c)$ on input $y = 2^{2^{x^c}}$. This simulation needs time $|y|^{c'} \leq \left(1 + 2^{x^c}\right)^{c'} \leq 2^{2c'x^c} \leq 2^{2c'(2^{|x|+1})^c} \leq 2^{2^{d|x|}}$ for some constant d and sufficiently large x.

Analogously one shows (2.2) and (2.3). We now prove the four statements.
1. If EE $\neq$ NEE, then let $L \in \mathrm{NEE} - \mathrm{EE}$. Choose $c \geq 1$ such that $L \in \mathrm{NTIME}(2^{2^{c \cdot n}})$. By (2.1) and (2.2), $B(L,c) \in \mathrm{NP} - \mathrm{P}$.
2. If EE $\neq$ NEE $\cap$ coNEE, then let $L \in (\mathrm{NEE} \cap \mathrm{coNEE}) - \mathrm{EE}$. Choose $c \geq 1$ such that $L, \overline{L} \in \mathrm{NTIME}(2^{2^{c \cdot n}})$. By (2.1)–(2.3), $B(L,c) \in (\mathrm{NP} \cap \mathrm{coNP}) - \mathrm{P}$.
3. If NEE $\neq$ coNEE, then let $L \in \mathrm{NEE} - \mathrm{coNEE}$. Choose $c \geq 1$ such that $L \in \mathrm{NTIME}(2^{2^{c \cdot n}})$. By (2.2) and (2.3), $B(L,c) \in \mathrm{NP} - \mathrm{coNP}$.
4. Let $L \in (\mathrm{NEEE} \cap \mathrm{coNEEE}) - (\mathrm{UEEE} \cap \mathrm{coUEEE})$ and choose some $c \geq 1$ such that L (resp., $\overline{L}$) is decidable by a nondeterministic machine N (resp, $\overline{N}$) that works in time $2^{2^{2^{c \cdot n}}}$. Let $B = \{t(c \cdot |x|) + x \mid x \in L\}$ and note that $B \subseteq \{t(c \cdot i) + k \mid i \in \mathbb{N}, 0 \leq k < 2^i\} \subseteq \{t(i) + k \mid i \in \mathbb{N}, 0 \leq k < 2^i\}$. We show $B \in (\mathrm{NP} \cap \mathrm{coNP}) - (\mathrm{UP} \cap \mathrm{coUP})$: $B \in \mathrm{NP}$ by the algorithm that on input $y = t(c \cdot |x|) + x$ simulates N on x in nondeterministic polynomial time in $|y|$ (N on x needs time $2^{2^{2^{c \cdot |x|}}} = \log t(c \cdot |x|) \leq \log y \leq |y|$). Similarly, $\overline{B} \in \mathrm{NP}$ by the algorithm that on input $y = t(c \cdot |x|) + x$ simulates N' on x in nondeterministic polynomial time in $|y|$. $B \notin (\mathrm{UP} \cap \mathrm{coUP})$, since otherwise $L \in \mathrm{UEEE} \cap \mathrm{coUEEE}$ by the algorithm that on input x simulates the $(\mathrm{UP} \cap \mathrm{coUP})$-algorithm for B on input $y = t(c \cdot |x|) + x$ (the simulation works in time $|y|^{c'} \leq (1 + \log y)^{c'} \leq (\log y)^{c'+1} \leq (\log 2t(c \cdot |x|))^{c'+1} = (2^{2^{2^{c \cdot |x|}}} + 1)^{c'+1} \leq (2^{2^{2^{c \cdot |x|}}})^{c'+2} \leq 2^{2^{2^{c'' \cdot |x|}}}$. □

2.4.7 Relativization

Modular algorithms combine different subroutines to complete their task, and sometimes, we would like to measure the performance of an algorithm while disregarding the resources needed by the subroutines. This scenario is modeled in complexity theory by oracle Turing machines and relativized complexity classes.

For the purpose of this thesis, oracles are always partial functions. The usually considered case of sets as oracles is included as a special case, when a set is identified by its characteristic function. An *oracle Turing machine* M equipped with the oracle g contains a write-only oracle input tape, a separate read-only oracle output tape, and a special oracle query state q. When M enters the state q with the string x currently on the oracle input tape and $g(x)$ is defined, then $g(x)$ immediately appears on the oracle output tape. If $g(x)$ is not defined, then the special symbol $\perp$ appears on the oracle output tape.

If M is nondeterministic, it has to act deterministically as long as the oracle input tape is not empty. If M is a transducer, then let M^g denote the partial function computed by M with oracle g. Otherwise, let $L(M^g)$ denote the language decided or accepted by M with oracle g.

We obtain *relativized complexity classes* by imposing resource bounds on the machines. Note that strictly, only complexity classes defined by machines can be relativized and care has to be taken in how this relativization is obtained exactly. We define relativized deterministic time classes as an example.

Definition 2.13. Let $r\colon \mathbb{N} \to \mathbb{N}$ be some monotonically increasing function, g a partial function and $\mathcal{C}$ an arbitrary complexity class.

$$(\mathrm{DTIME}(r))^g = \{A \subseteq \mathbb{N} \mid \text{there is a deterministic Turing machine } M \text{ that accepts } A \text{ in time } r \text{ with oracle } g\}$$
$$(\mathrm{DTIME}(r))^{\mathcal{C}} = \bigcup_{O \in \mathcal{C}} (\mathrm{DTIME}(r))^O$$

Using relativization, one can define the so-called *polynomial-time hierarchy*.

Definition 2.14. Let $k \in \mathbb{N}$.

$$\Sigma_0^{\mathrm{P}} = \Delta_0^{\mathrm{P}} = \Pi_0^{\mathrm{P}} = \mathrm{P}$$
$$\Sigma_{k+1}^{\mathrm{P}} = \mathrm{NP}^{\Sigma_k^{\mathrm{P}}} \qquad \Delta_{k+1}^{\mathrm{P}} = \mathrm{P}^{\Sigma_k^{\mathrm{P}}} \qquad \Pi_k^{\mathrm{P}} = \mathrm{co}\Sigma_k^{\mathrm{P}}$$
$$\mathrm{PH} = \bigcup_{k \in \mathbb{N}} \Sigma_k^{\mathrm{P}}$$

An equivalent definition of the Σ_k^{P}- and Π_k^{P}-levels can be given using the projection operator, since it holds that $\Sigma_{k+1}^{\mathrm{P}} = \exists^{\mathrm{P}} \cdot \Pi_k^{\mathrm{P}}$ for every $k \in \mathbb{N}$.

It is an open question whether the polynomial-time hierarchy is infinite or not. If for some k, it holds that $\Sigma_k^{\mathrm{P}} = \mathrm{PH}$, one says that the polynomial-time hierarchy *collapses* to the kth level. Note that if $\mathrm{P} = \mathrm{NP}$, then $\mathrm{PH} = \mathrm{P}$, so the assumption that the polynomial-time hierarchy does not collapse is at least as strong as $\mathrm{P} \neq \mathrm{NP}$.

A hierarchy of a similar kind collapses for the class BPP:

Theorem 2.15 ([Ko82])**.** $\mathrm{BPP}^{\mathrm{BPP}} = \mathrm{BPP}$

Complexity classes $\mathcal{C}$ that are defined via machines and allow relativization and have the property that $\mathcal{C}^{\mathcal{C}} = \mathcal{C}$ are called *self-low*.

Relativization can also be used to show the limits of current proof techniques. Almost all known proofs about complexity classes are *relativizing*, which means that they remain valid in a setting where each machine is equipped with the same oracle. For example, one can show that $\mathrm{P}^A \subseteq \mathrm{NP}^A$ holds for all $A \subseteq \mathbb{N}$ with only minor changes to the proof for $\mathrm{P} \subseteq \mathrm{NP}$. Baker, Gill and Solovay showed that no relativizing proof can be used to establish $\mathrm{P} = \mathrm{NP}$ or $\mathrm{P} \neq \mathrm{NP}$. Similar results are true for many open questions in complexity theory.

Theorem 2.16 ([BGS75])**.** There are $A, B \subseteq \mathbb{N}$ such that $\mathrm{P}^A = \mathrm{NP}^A$ and $\mathrm{P}^B \neq \mathrm{NP}^B$.

2.4.8 Reducibilities

Different sets and functions are compared with respect to their computability or complexity using *reductions*. The general concept is that A reduces to B if and only if A can be solved with the help of a (perhaps hypothetical) subroutine that solves B. This means that B is at least as hard to solve as A. The most general reduction considered is the (unbounded) Turing reduction, which is defined using oracle Turing machines.

Definition 2.17. A partial function (or set represented by its characteristic functions) f is *unboundedly Turing reducible* to a partial function (or set) g, in symbols $f \leq_{\mathrm{T}} g$ if $f = M^g$ for some Turing machine M.

This reduction is only useful for complexity theory, when the power of the reducing machine is restricted, for example to polynomial time (again, the following definition also applies to sets):

Definition 2.18. A partial function f is *(polynomial-time) Turing reducible* to a partial function g, in symbols $f \leq_{\mathrm{T}}^{\mathrm{p}} g$ if $f = M^g$ for some polynomial-time Turing machine M.

Similarly, one defines *logarithmic-space Turing reduction*, $\leq_{\mathrm{T}}^{\log}$ using a Turing machine with logarithmic space bound. The more restrictive *many-one reductions* and *truth-table reductions* are not defined via oracles (although they can) and are only considered for sets, here.

Definition 2.19. A set $A \subseteq \mathbb{N}$ *(unboundedly) many-one reduces* to a set $B \subseteq \mathbb{N}$, $A \leq_{\mathrm{m}} B$, if there is some recursive function f such that for all $x \in \mathbb{N}$ it holds that $x \in A \iff f(x) \in B$.

Definition 2.20. A set $A \subseteq \mathbb{N}$ *polynomial-time many-one reduces* to a set $B \subseteq \mathbb{N}$, $A \leq_{\mathrm{m}}^{\mathrm{p}} B$, if there is some $f \in \mathrm{FP}$ such that for all $x \in \mathbb{N}$ it holds that $x \in A \iff f(x) \in B$.

Analogously, one defines the *logarithmic-space many-one reduction*, $\leq_{\mathrm{m}}^{\log}$, which will be the default reduction we use for comparing the complexity of sets. Less common reductions are the following:

Definition 2.21. A set $A \subseteq \mathbb{N}$ *conjunctively truth-table reduces in polynomial time* to a set $B \subseteq \mathbb{N}$, $A \leq_{\mathrm{ctt}}^{\mathrm{p}} B$, if there is a polynomial-time computable function $f\colon \mathbb{N} \to 2^{\mathbb{N}}$ such that for all $x \in \mathbb{N}$ it holds that $x \in A \iff f(x) \subseteq B$.

The *polynomial-time disjunctive truth-table reduction*, $\leq_{\mathrm{dtt}}^{\mathrm{p}}$ is defined analogously, where the condition is that $x \in A \iff f(x) \cap B \neq \emptyset$.

These two reductions can be combined to obtain the $\leq_{\mathrm{dtt(ctt)}}^{\mathrm{p}}$-reduction: It holds that $A \leq_{\mathrm{dtt(ctt)}}^{\mathrm{p}} B$ for two sets $A, B \subseteq \mathbb{N}$ if there is a polynomial-time computable function f whose output is a set of sets of natural numbers such that for all $x \in \mathbb{N}$ it holds that $x \in A \iff \exists Q \in f(x)\colon Q \subseteq B$.

Observe that if only partial functions with polynomially bounded output length are considered (this of course includes sets), then each reduction relation defined above is reflexive and transitive (this is not always immediately clear). With each reduction $\leq_{\mathrm{Y}}^{X}$ we associate an equivalence relation $\equiv_{\mathrm{Y}}^{X}$ in the usual way. An equivalence class with respect to this relation is called a *degree*.

Definition 2.22. Let $\mathcal{C}$ be a complexity class. We say that $\mathcal{C}$ is *closed* under a reduction $\leq_Y^X$ (or also $\leq_Y^X$-*closed*) if for all $A \in \mathcal{C}$ and all $B \subseteq \mathbb{N}$ such that $B \leq_Y^X A$, we have $B \in \mathcal{C}$. A set $A \subseteq \mathbb{N}$ is $\leq_Y^X$-*hard* for $\mathcal{C}$ if for all $B \in \mathcal{C}$ we have $B \leq_Y^X A$. The set A is called $\leq_Y^X$-*complete* for $\mathcal{C}$ if we additionally have $A \in \mathcal{C}$.

If the reduction is not explicitly given in statements about hardness or completeness, we assume the $\leq_{\mathrm{m}}^{\log}$ reduction for sets and for function classes, we instead use the $\leq_{\mathrm{T}}^{\mathrm{p}}$-reduction.

Property 2.23. The classes NP and coNP are closed under $\leq_{\mathrm{dtt}}^{\mathrm{p}}$-, $\leq_{\mathrm{ctt}}^{\mathrm{p}}$- and $\leq_{\mathrm{dtt(ctt)}}^{\mathrm{p}}$-reductions.

Proof. For NP, assume that $A \leq_{\mathrm{dtt}}^{\mathrm{p}} B \in \mathrm{NP}$ via f. On input x, compute $f(x)$ and simulate the nondeterministic polynomial-time machine for B on each element in $f(x)$ one after another and accept if any of the simulations accept. The proof is analogous for the other cases. □

2.4.9 NP-complete problems

One of the first non-artificial problems for which NP-completeness could be shown is the problem of determining whether a Boolean formula is satisfiable or not. More formally, this problem is defined as follows, where we restrict to formulas in conjunctive normal form:

Definition 2.24. Let $x_1, x_2, \ldots$ be Boolean variables. A variable or a negated variable $\neg x_i$ is called a *literal*. A set of finitely many literals is called a *clause*. A set of clauses is called a *Boolean formula in conjunctive normal form* (CNF). An assignment of Boolean values to each of the variables can make the formula satisfied: A variable is satisfied if it is assigned the value true, and a negated variable is satisfied if it is assigned the value false. A clause is satisfied if any of its literals is satisfied and a Boolean formula in CNF is satisfied if all of its clauses are satisfied. Furthermore, a Boolean formula in CNF is *satisfiable* if there is an assignment to its variables such that the formula is satisfied.

$$\textsc{Sat} = \{\langle f \rangle \mid f \text{ is a Boolean formula in CNF that is satisfiable}\}$$

If one restricts the number of literals per clause in a Boolean formula in CNF to exactly three, one obtains a formula in 3-CNF and the respective satisfiability problem, 3Sat is still NP-complete. This property makes this problem convenient to show that other problems are NP-hard via reductions.

Another NP-complete problem is the Knapsack problem. Here, one is given a set of objects with weights and values and has to decide if there is a subset that respects a given restriction about the sum of the weights and the sum of the values.

Definition 2.25.

$$\textsc{Knapsack} = \{\langle w, v, k, \langle w_0, \ldots, w_k \rangle, \langle v_0, \ldots, v_k \rangle\rangle \mid \exists I \subseteq \{0, 1, \ldots, k\}: \sum_{i \in I} w_i \leq w \wedge \sum_{i \in I} v_i \geq v\}$$

Interestingly, a natural view on the knapsack problem is that of a two-objective optimization problem, where one has to maximize the values of the chosen objects and at the same time minimize their weight.

2.4.10 Approximability

Here, we give relevant definitions for the area of (single-objective) approximation algorithms. These will all be extended to multiobjective optimization problems in the main part of the thesis.

Definition 2.26. A (single-objective) *optimization problem* is a tuple $\mathcal{O} = (S, f, d)$, where

- $S\colon \mathbb{N} \to 2^{\mathbb{N}}$ is a total function that maps a problem instance to the set of *feasible solutions*, there is a polynomial p such that for each $s \in S(x)$ it holds that $s < 2^{p(|x|)}$ and $\{\langle x, s\rangle \mid s \in S(x)\} \in \mathrm{P}$.
- $f \in \mathrm{FP}$ is a total function that assigns a solution its value from $\mathbb{N}$ depending also on the problem instance.
- $d \in \{\min, \max\}$ specifies the direction of optimization.

Definition 2.27. Let $\mathcal{O} = (S, f, d)$ be an optimization problem. A function $g\colon \mathbb{N} \to \mathbb{N}$ *solves* the problem $\mathcal{O}$ if for each $x \in \mathbb{N}$ it holds that: If $S(x) = \emptyset$ then $g(x)$ is undefined and otherwise $g(x) \in S(x)$ and $f(\langle x, g(x)\rangle) \geq f(\langle x, s\rangle)$ (if $d = \max$) or $f(\langle x, g(x)\rangle) \leq f(\langle x, s\rangle)$ (if $d = \min$) for every $s \in S(x)$. $\mathcal{O}$ is *solvable in polynomial time* if there is a function $g \in \mathrm{FP}$ that solves $\mathcal{O}$.

Definition 2.28. Let $\mathcal{O} = (S, f, d)$ be an optimization problem and $\alpha \in \mathbb{R}$, $\alpha > 1$. A function $g\colon \mathbb{N} \to \mathbb{N}$ *approximately solves* the problem $\mathcal{O}$ *with approximation ratio* α (or *is an* α*-approximation for* $\mathcal{O}$) if for each $x \in \mathbb{N}$ it holds that: If $S(x) = \emptyset$ then $g(x)$ is undefined and otherwise $g(x) \in S(x)$ and $\alpha f(\langle x, g(x)\rangle) \geq f(\langle x, s\rangle)$ (if $d = \max$) or $f(\langle x, g(x)\rangle) \leq \alpha f(\langle x, s\rangle)$ (if $d = \min$) for every $s \in S(x)$. $\mathcal{O}$ is *α-approximable in polynomial time* if there is a function $g \in \mathrm{FP}$ that approximately solves $\mathcal{O}$ with approximation ratio α. It is *α-approximable in randomized polynomial time* if there is a function g that can be computed in randomized polynomial time and approximately solves $\mathcal{O}$ with approximation ratio α with probability at least $1/2$ for each instance.

Definition 2.29. A function f is a *PTAS* (*polynomial-time approximation scheme*) for an optimization problem $\mathcal{O}$ if for each $k \geq 1$, the function $x \mapsto f(2^k, x) \in \mathrm{FP}$ is a $(1 + \frac{1}{k})$-approximation for $\mathcal{O}$. It is an *FPTAS* (*fully polynomial-time approximation scheme*) if additionally $f \in \mathrm{PF}$. The notions *PRAS* (*polynomial-time randomized approximation scheme*) and *FPRAS* (*fully polynomial-time randomized approximation scheme*) are defined analogously.

Definition 2.30. Let APX be the class of optimization problems $\mathcal{O}$ where there exists an $\alpha > 1$ such that $\mathcal{O}$ has an α-approximation.

The problem 3Sat defined above asks if on input of a Boolean formula in 3-CNF there is some assignment to the variables such that all clauses are satisfied. In the following optimization problem, the task is to simply satisfy as much clauses as possible:

Definition 2.31. Let Max3Sat $= (S, f, \max)$, where we assume that instances are bijectively encoded as Boolean formulas in 3-CNF and

- $S(\langle\varphi\rangle) = \{\langle a_1, \ldots, a_k\rangle \mid \varphi \text{ has } k \text{ variables and } a_1, \ldots, a_k \in \{0, 1\}\}$

- $f(\langle\langle\varphi\rangle, \langle a_1, \ldots, a_k\rangle\rangle) =$ number of clauses in φ satisfied by the assignment $x_i = a_i$

We know that MAX3SAT is in APX by a simple idea: On input of a Boolean formula φ consider some arbitrary assignment I of the variables and its inverted assignment I^*. Observe that any clause of φ is satisfied by I or by I^*. So at least one of I or I^* satisfies at least one half of all clauses. Thus, we arrive at a 2-approximation of MAX3SAT. It can be shown that MAX3SAT does not have a PTAS unless P = NP [AL97].

2.5 Computability Theory

A set $A \subseteq \mathbb{N}$ is *(computably) enumerable* or *(recursively) enumerable* if it can be accepted by a Turing machine. The class of these sets is abbreviated by RE. If the characteristic function c_A of a set A is computable, the set A is called *decidable* or *recursive*. The class of these sets is abbreviated as REC.

These classes are relativized as follows: For $O \subseteq \mathbb{N}$, the class REC^O contains all sets $A \subseteq \mathbb{N}$ such that c_A is computable by a Turing machine with oracle access to O. The class RE^O contains all sets $A \subseteq \mathbb{N}$ that can be accepted by a Turing machine with oracle access to O.

Using these relativized classes, the *arithmetical hierarchy* can be defined similarly to the already mentioned polynomial-time hierarchy (though its definition predates the one for the polynomial-time hierarchy):

Definition 2.32. Let $k \in \mathbb{N}$.

$$\begin{array}{ccc} & \Sigma_0 = \Delta_0 = \Pi_0 = \mathrm{REC} & \\ \Sigma_{k+1} = \mathrm{RE}^{\Sigma_k} & \Delta_{k+1} = \mathrm{REC}^{\Sigma_k} & \Pi_k = \mathrm{co}\Sigma_k \\ & \mathrm{AH} = \bigcup_{k\in\mathbb{N}} \Sigma_k & \end{array}$$

Again, an equivalent definition of the Σ_k- and Π_k-levels can be given using a projection operator, which needs to be unbounded now: For any $A \in \mathbb{N}$ define the *unbounded projection* of A as $\exists\cdot A = \{x \in \mathbb{N} \mid \exists y\colon \langle x, y\rangle \in A\}$ and extend this operator to classes by $\exists\cdot\mathcal{C} = \bigcup_{A\in\mathcal{C}} \exists\cdot A$. It holds that $\Sigma_{k+1} = \exists^{\mathrm{p}}\cdot\Pi_k$.

For the arithmetical hierarchy, it is known that it is infinite and also that $\Delta_k = \Sigma_{k+1} \cap \Pi_{k+1}$, which is an open question for the polynomial-time hierarchy.

Definition 2.33. There are special so-called *universal Turing machines* M that can simulate all other machines in the sense that for every Turing machine M' there is some $i \in \mathbb{N}$ such that $M(\langle i, x\rangle) = M'(x)$ for all $x \in \mathbb{N}$. If we further require for M that there is a recursive function $s\colon \mathbb{N} \to \mathbb{N}$ such that for all $i, x, y \in \mathbb{N}$ it holds that $M(\langle s(\langle i, x\rangle), y\rangle) = M(\langle i, \langle x, y\rangle\rangle)$ we can use M to obtain an enumeration of all Turing machines and thus of all computable functions and enumerable sets by $M_i\colon \mathbb{N} \to \mathbb{N}, x \mapsto M(\langle i, x\rangle)$.

Note that because of the additional requirement it is not important which universal Turing machine is used here as the indices i can be effectively transformed from one enumeration to another one. To see this, assume that M_1 provides one enumeration

$\{M_{1,i}\}$ and M_2 provides a second one $\{M_{2,i}\}$. Let $m_2 \in \mathbb{N}$ such that $M_2 = M_{1,m_2}$. For an arbitrary Turing machine $M = M_{2,i}$ and any $x \in \mathbb{N}$, we have

$$M(x) = M_2(\langle i, x\rangle) = M_1(\langle m_2, \langle i, x\rangle\rangle) = M_1(\langle s(\langle m_2, i\rangle), x\rangle) = M_{1,s(\langle m_2,i\rangle)}(x)$$

and thus $s(\langle m_2, i\rangle)$ is the index number of M in the enumeration defined by M_1.

Because of the above arguments, we can simply fix an arbitrary universal Turing machine and its respective enumeration $\{M_i\}$ for the rest of this thesis. Many basic problems in computability theory are concerned with such enumerations. Perhaps the most fundamental problem is the halting problem.

Definition 2.34. The *halting problem* is the set $\mathrm{K} = \{\langle i, x\rangle \mid M_i(x) \text{ is defined}\}$.

Proposition 2.35. K is $\leq_{\mathrm{m}}^{\log}$-complete for RE.

Since no decidable set can be $\leq_{\mathrm{m}}^{\log}$-hard for RE, we can deduce that it is in general not possible to decide if a given Turing machine will halt on a given input or not.

Chapter 3

Complexity of Search and Decision Problems

As a theoretical foundation for the systematic study of multiobjective optimization problems, we first turn to the relation between search problems (sometimes called relation problems) and decision problems in general. Decision problems are the problems that are classically considered in complexity theory: A *decision problem* A is a subset of $\mathbb{N}$ (sometimes, subsets of Σ^* for a finite alphabet Σ are considered, but this does not make any difference) and the associated computational task is to compute its characteristic function c_A, i. e. on input $x \in \mathbb{N}$, determine if $x \in A$ or not. Examples for decision problems are the set of primes $\text{PRIMES} = \{x \in \mathbb{N} \mid x \text{ is prime}\}$, the set of satisfiable Boolean formulas $\textsc{Sat} = \{\langle\varphi\rangle \in \mathbb{N} \mid \varphi \text{ is a Boolean formula in } k \text{ variables and there are } a_1, \ldots, a_k \in \{0,1\} \text{ such that } \varphi(a_1, \ldots, a_k) = 1\}$. Decision problems can be summarized as problems where the question is asked if a given object has some property or not. *Search problems*, in contrast, are more complicated. We refer to the next section for a formal definition of the search problems considered here, but in general, the task is to decide if a given object has a specific property and, if this is the case, compute some witness. As an example, the search problem usually associated with the decision problem Sat is the task of deciding whether some given Boolean formula φ is satisfiable and additionally computing one of its satisfying assignments. The essential difference now is that there is exactly one correct answer when solving a decision problem but there can be several correct answers when solving a search problem. As a multiobjective problem can also have multiple optimal solutions for each instance, search problems will turn out suitable for formalizing multiobjective optimization problems.

We begin this chapter by formalizing search problems as multivalued functions and defining important classes of multivalued functions. Furthermore, we define a suitable reduction relation between multivalued functions that is a generalization of the usual Turing-reduction between decision problems. As comparing decision and search problems by equality is not very useful, we will compare them with respect to the reduction we defined and also extend this to classes of decision and search problems. Note that previous research mainly focused on comparing classes of search problems or classes of multivalued functions with respect to equality and refinements. By comparing classes with respect to Turing reductions, one obtains more robust results and is able to compare decision and search problems.

We will study the relations between various classes of search and decision problems, while we specifically focus on classes relevant for multiobjective optimization. In most cases where we are not able to show that an inclusion (with respect to Turing-reduction) holds, we show that such an inclusion would have implausible complexity-theoretic consequences. Figure 3.1 summarizes the inclusions obtained in this chapter. In chapter 5, we will add several classes relevant for multiobjective optimization to this diagram and obtain Figure 5.3 on page 80.

The main result here is that the intuition that search problems are different from decision problems because of the fact that they can have more than one correct solution is right: We show that (under some assumption about super-exponential time classes) there are (NP) search problems that are Turing-inequivalent to any decision problem of arbitrary complexity (Corollary 3.22). In contrast, any search problem that has at most one correct answer for each instance (so-called singlevalued functions) is equivalent to a decision problem (Proposition 3.15). Beame et al. [BCE+98] have shown that the first result also holds relative to a generic oracle, which further supports our claim that search and decision problems are fundamentally different.

3.1 Definitions and First Results

We use (polynomially bounded) multivalued functions to model search problems. Book, Long and Selman [BLS84] used nondeterministic polynomial-time machines that can generate outputs on each computation path to define NPMV, an important class of multivalued functions. Recently, an approach similar to the one used to define NP sets as polynomially bounded projections of sets in P has become popular which can also be used to define other interesting classes of multivalued functions in the same way.

Definition 3.1. A *multivalued function* is a total function $\mathbb{N} \to 2^{\mathbb{N}}$. For a multivalued function f, define its *domain*, *graph* and *range* as $\mathrm{dom}(f) = \{x \in \mathbb{N} \mid f(x) \neq \emptyset\}$, $\mathrm{graph}(f) = \{(x, y) \in \mathbb{N}^2 \mid y \in f(x)\}$, and $\mathrm{range}(f) = \bigcup_{x \in \mathbb{N}} f(x)$. If f is viewed as a multivalued function from A to B, it is called *total* if $\mathrm{dom}(f) = \mathbb{N}$. A multivalued function g is a *refinement* of a multivalued function f, if $\mathrm{dom}(g) = \mathrm{dom}(f)$ and for all x, $g(x) \subseteq f(x)$. For two classes of multivalued functions $\mathcal{C}$ and $\mathcal{D}$ we say that $\mathcal{C}$ *has refinements in* $\mathcal{D}$, $\mathcal{C} \subseteq_{\mathrm{c}} \mathcal{D}$, if for every $f \in \mathcal{C}$ there is some $g \in \mathcal{D}$ such that g is a refinement of f.

To get a unified view on multivalued functions, ordinary partial functions and sets, we sometimes interpret a partial function $g\colon \mathbb{N} \to \mathbb{N}$ as the multivalued function $g'\colon x \mapsto \{g(x) \mid g(x) \text{ defined}\}$ and a set $A \subseteq \mathbb{N}$ as its characteristic function c_A. This enables us to generalize the ordinary $\leq^{\mathrm{p}}_{\mathrm{T}}$-reduction to multivalued functions.

Definition 3.2 ([FHOS97])**.**

1. Recall Definition 2.18: Let f and g be partial functions. *f is polynomial-time Turing reducible to g*, $f \leq^{\mathrm{p}}_{\mathrm{T}} g$, if there exists a deterministic, polynomial-time oracle Turing machine M such that $f = M^g$.

2. Let f and g be multivalued functions. *f is polynomial-time Turing reducible to g*, $f \leq^{\mathrm{p}}_{\mathrm{T}} g$, if there exists a deterministic, polynomial-time oracle Turing machine M

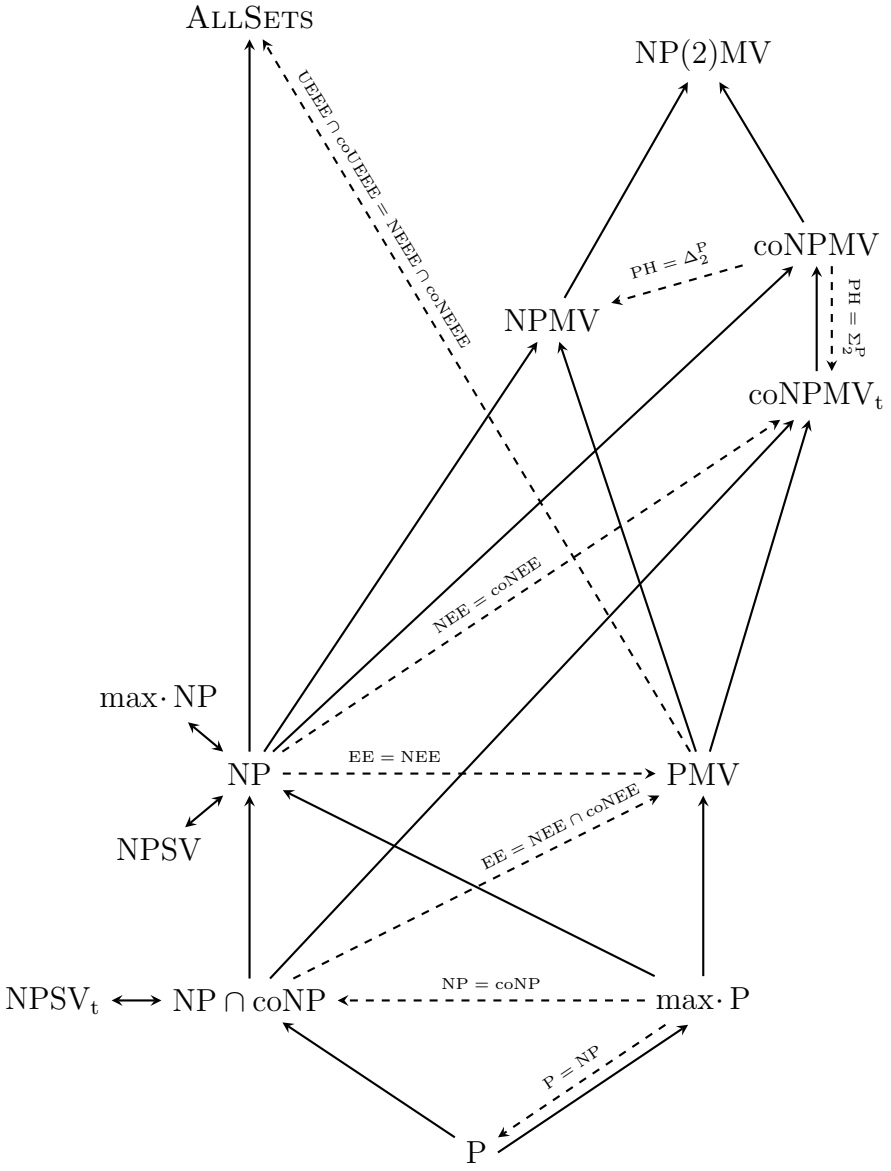


Figure 3.1: Summary of embeddings of complexity classes. A (non-dashed) arrow from $\mathcal{C}$ to $\mathcal{D}$ denotes that $\mathcal{C}$ can be embedded in $\mathcal{D}$, i. e. $\forall f \in \mathcal{C}\, \exists g \in \mathcal{D}\colon f \equiv_T^P g$. Dashed arrows give evidence against such an embedding (the embedding implies the arrow's label).
Observe that the embedding relation is reflexive and transitive and that evidence against an embedding propagates along non-dashed arrows (heads of dashed arrows can be moved downwards, tails can be moved upwards). Thus, we provide evidence against all embeddings that are not explicitly shown to hold except for NPMV $\subseteq_{\equiv_T^P}$ coNPMV, NP(2)MV $\subseteq_{\equiv_T^P}$ coNPMV, coNPMV$_t$ $\subseteq_{\equiv_T^P}$ NPMV and the embedding relations between P, NP $\cap$ coNP and NP (obviously).
Note that PMV = NPMV$_g$, max$\cdot$ NP = OptP (Krentel [Kre88]) and AllSets is the class of all decision problems. We will extend this diagram by classes of functions relevant in multiobjective optimization in Figure 5.3 on page 80.

such that for every partial function g' that is a refinement of g it holds that the partial function $M^{g'}$ is a refinement of f.

It is important to note that the definition above is different from the one given in an article by Selman [Sel94]. In Selman's definition, if the oracle g is a multivalued function and if some q with $g(q) = \emptyset$ is queried, then the oracle can give an arbitrary answer. Also note that the oracle model described above ensures that $\leq^{\mathrm{p}}_{\mathrm{T}}$ is reflexive and transitive.

We also define the relation $\equiv^{\mathrm{p}}_{\mathrm{T}}$ of *polynomial-time Turing-equivalence* as usual. By interpreting a set $A \subseteq \mathbb{N}$ as its characteristic function c_A, the definition above also applies to sets and in fact coincides with the usual polynomial-time Turing reduction in that case.

A multivalued function g is called *polynomial-time solvable*, if there is a polynomial-time computable, partial function f such that f is a refinement of g. A multivalued function g is called NP-*hard*, if all problems in NP are polynomial-time Turing-reducible to g.

We now introduce the operators $\mathrm{wit}\cdot$ and $\max\cdot$ similarly to $\exists^{\mathrm{p}}\cdot$. They will help us to define classes of problems and multivalued functions.

Definition 3.3. For a set $A \subseteq \mathbb{N}$ and a total function $p\colon \mathbb{N} \to \mathbb{N}$ (usually a polynomial) we define the multivalued function $\mathrm{wit}_p\cdot A$ from $\mathbb{N}$ to $\mathbb{N}$ and the total function $\max_p\cdot A\colon \mathbb{N} \to \mathbb{N}$ as follows:

$$\begin{aligned} \mathrm{wit}_p\cdot A\colon x &\mapsto \{y \in \mathbb{N} \mid \langle x, y\rangle \in A \text{ and } y < 2^{p(|x|)}\} \\ \max_p\cdot A\colon x &\mapsto \max(\{0\} \cup \mathrm{wit}_p\cdot A(x)) \end{aligned}$$

Moreover, for a set $A \subseteq \mathbb{N}$ and a complexity class $\mathcal{C} \subseteq 2^{\mathbb{N}}$ we define:

$$\begin{aligned} \mathrm{wit}\cdot A &= \{\mathrm{wit}_p\cdot A \mid p \text{ is a polynomial}\} \\ \max\cdot A &= \{\max_p\cdot A \mid p \text{ is a polynomial}\} \\ \mathrm{wit}\cdot \mathcal{C} &= \bigcup_{A\in\mathcal{C}} \mathrm{wit}\cdot A \\ \max\cdot \mathcal{C} &= \bigcup_{A\in\mathcal{C}} \max\cdot A \end{aligned}$$

If p is a polynomial, then the task of the search problem $\mathrm{wit}_p\cdot A$ is to compute an arbitrary so-called *witness* from the witness set A relative to the decision problem $\exists_p\cdot A = \mathrm{dom}(\mathrm{wit}_p\cdot A)$. The task of $\max_p\cdot A$ is to compute the largest witness. The elements from $\mathrm{wit}\cdot \mathrm{P}$ are sometimes called NP *search problems*.

This unified approach in defining search problems was also undertaken by Große and Hempel [GH03]. Their operator $\mathrm{rel}\cdot$, which was already defined by Wechsung [Wec00], is identical to $\mathrm{wit}\cdot$. Classes like $\max\cdot \mathrm{P}$ and $\max\cdot \mathrm{NP}$ were systematically investigated by Hempel and Wechsung [HW00]. Note that $\max\cdot \mathrm{NP}$ is identical to Krentel's OptP [Kre88] and is one way to formalize single-objective optimization problems. For special complexity classes $\mathcal{C}$, the functions in $\mathrm{wit}\cdot \mathcal{C}$ are well-known: The classes $\mathrm{wit}\cdot \mathrm{P}$, $\mathrm{wit}\cdot \mathrm{NP}$, and $\mathrm{wit}\cdot \mathrm{coNP}$ were studied under the names $\mathrm{NPMV_g}$, NPMV, and coNPMV by Selman [Sel92, Sel94, Sel96], Fenner et al. [FHOS97, FGH$^+$99], and Hemaspaandra et al. [HNOS96].

Definition 3.4 ([BLS84, Sel92, HNOS96])**.** Let N be a nondeterministic Turing machine that has an output tape. The machine N *computes the multivalued function* f from $\mathbb{N}$ to

$\mathbb{N}$ where

$$f(x) = \{y \in \mathbb{N} \mid \text{some accepting computation path of } N \text{ on input } x \text{ contains the binary representation of } y \text{ on the output tape}\}.$$

We now state the usual definitions of important classes of these functions which we generalize later.

$$\begin{aligned}
\mathrm{NPMV} &= \{f \mid f \text{ is computed by some nondeterministic polynomial-time Turing machine}\} \\
\mathrm{NPMV_g} &= \{f \in \mathrm{NPMV} \mid \mathrm{graph}(f) \in \mathrm{P}\} \\
\mathrm{coNPMV} &= \{f \mid f \text{ is a multivalued function from } \mathbb{N} \text{ to } \mathbb{N} \text{ and there is a polynomial } p \text{ such that } f(x) < 2^{p(|x|)} \text{ and } \mathrm{graph}(f) \in \mathrm{coNP}\}
\end{aligned}$$

Proposition 3.5.

1. $\mathrm{wit}\cdot \mathrm{P} = \mathrm{NPMV_g}$.
2. $\mathrm{wit}\cdot \mathrm{NP} = \mathrm{NPMV}$.
3. $\mathrm{wit}\cdot \mathrm{coNP} = \mathrm{coNPMV}$.

The name NPMV stands for *nondeterministic polynomial-time multivalued*. We will adopt this general naming scheme for multivalued functions:

Definition 3.6. For any class of decision problems $\mathcal{C} \subseteq 2^{\mathbb{N}}$, we define the related class of *multivalued functions* $\mathcal{C}\mathrm{MV}$, *total multivalued functions* $\mathcal{C}\mathrm{MV_t}$, *singlevalued functions* $\mathcal{C}\mathrm{SV}$ and *total singlevalued functions* $\mathcal{C}\mathrm{SV_t}$ as follows:

$$\begin{aligned}
\mathcal{C}\mathrm{MV} &= \mathrm{wit}\cdot \mathcal{C} \\
\mathcal{C}\mathrm{MV_t} &= \{f \in \mathcal{C}\mathrm{MV} \mid \mathrm{dom}(f) = \mathbb{N}\} \\
\mathcal{C}\mathrm{SV} &= \{f \in \mathcal{C}\mathrm{MV} \mid \forall x\colon \#(f(x)) \leq 1\} \\
\mathcal{C}\mathrm{SV_t} &= \mathcal{C}\mathrm{SV} \cap \mathcal{C}\mathrm{MV_t}
\end{aligned}$$

Note that a multivalued function f with polynomially bounded values is in $\mathcal{C}\mathrm{MV}$ if and only if $\mathrm{graph}(f) \in \mathcal{C}$. Furthermore, we have $\mathrm{PMV} = \mathrm{NPMV_g}$ and will generally prefer the former name.

We will consider the following important example of a multivalued function or search problem:

Definition 3.7. We define the multivalued function

$$\text{f-}\textsc{Sat}(\varphi) = \{\langle x_1, \ldots, x_n\rangle \mid \varphi \text{ is a Boolean formula with exactly } n \text{ variables and } \varphi(x_1, \ldots, x_n) = 1\}.$$

Observe that f-$\textsc{Sat} \in \mathrm{PMV}$ and $\mathrm{dom}(\text{f-}\textsc{Sat}) = \textsc{Sat}$. A multivalued function with these properties is called an NP search problem for the decision problem $\textsc{Sat}$. The function f-$\textsc{Sat}$ is generally regarded as the canonical search problem for $\textsc{Sat}$ but of course it is not the only one. Furthermore, there are multivalued functions f of arbitrarily high complexity such that $\mathrm{dom}(f) = \textsc{Sat}$. It is obviously the case that $\textsc{Sat} \leq^{\mathrm{p}}_{\mathrm{T}} \text{f-}\textsc{Sat}$, but we also know

that f-SAT $\leq^{\mathrm{p}}_{\mathrm{T}}$ SAT holds, i. e., one can compute a satisfying assignment for a Boolean formula if one has access to an oracle that decides if a Boolean formula is satisfiable or not. We will get back to this property, called *search reduces to decision* [BD76, BBFG91] again in sections 3.4 and 5.1.

From the following proposition, it can be seen that PMV is not closed under $\leq^{\mathrm{p}}_{\mathrm{T}}$ (unless P = NP). In fact, the reduction closure of a fixed multivalued function f always contains multivalued functions g with arbitrarily complex graphs since it suffices to compute a simple refinement of g in the reduction. This means that none of the classes of multivalued functions we consider here are closed under $\leq^{\mathrm{p}}_{\mathrm{T}}$. This is of course not a problem, since we are normally concerned with Turing-equivalence only.

Proposition 3.8. The multivalued function f-SAT is $\leq^{\mathrm{p}}_{\mathrm{T}}$-hard for NPMV and thus $\leq^{\mathrm{p}}_{\mathrm{T}}$-hard for PMV.

Proof. Remember the proof for the NP-completeness of SAT. There, for an arbitrary $A \in \mathrm{NP}$, a nondeterministic polynomial-time Turing machine M accepting A and an input x is transformed to a Boolean formula φ that is satisfiable if and only if $x \in A$. More importantly, a satisfying assignments for φ can be obtained in polynomial time from an accepting computation path of M on x and vice-versa. This means that with oracle access to SAT we can obtain accepting computation paths of nondeterministic Turing machines. In other words, we have $g \leq^{\mathrm{p}}_{\mathrm{T}} \mathrm{SAT}$ for any $g \in \mathrm{wit}\cdot \mathrm{P} = \mathrm{PMV}$.

Let $f \in \mathrm{NPMV}$. Then there is a polynomial p and a set $B \in \mathrm{P}$ such that $f(x) = \{y \mid y < 2^{p(|x|)} \wedge \exists z < 2^{p(|x|)} \colon \langle x, y, z\rangle \in B\}$. For the function $g(x) = \{\langle y, z\rangle \mid y, z < 2^{p(|x|)}, \langle x, y, z\rangle \in B\} \in \mathrm{PMV}$ we obtain $f \leq^{\mathrm{p}}_{\mathrm{T}} g \leq^{\mathrm{p}}_{\mathrm{T}} \mathrm{SAT} \leq^{\mathrm{p}}_{\mathrm{T}}$ f-SAT and thus the assertion. □

We will see in Theorem 3.16 that f-SAT, and in fact any NPMV-function is not $\leq^{\mathrm{p}}_{\mathrm{T}}$-hard for coNPMV unless $\mathrm{PH} = \Delta^{\mathrm{P}}_2$.

3.2 Embeddings

Previous research on multivalued functions mainly focused on comparing classes of multivalued functions with respect to equality and refinements. By comparing classes with respect to Turing reductions, one obtains more robust results and is able to compare decision and search problems. As our main subject of investigation, we use the inclusion relation of $\leq^{\mathrm{p}}_{\mathrm{T}}$-degrees:

Definition 3.9. For two classes $\mathcal{C}, \mathcal{D}$ (of decision problems or multivalued functions), we say that $\mathcal{C}$ can be *embedded* in $\mathcal{D}$, denoted by $\mathcal{C} \subseteq_{\equiv^{\mathrm{p}}_{\mathrm{T}}} \mathcal{D}$, if and only if for each $f \in \mathcal{C}$ there is some $g \in \mathcal{D}$ such that $f \equiv^{\mathrm{p}}_{\mathrm{T}} g$. Two classes are *equivalent*, denoted by $\mathcal{C} =_{\equiv^{\mathrm{p}}_{\mathrm{T}}} \mathcal{D}$, if they can be embedded in each other.

We now want to systematically analyze the embeddings between some classes defined via the operators wit$\cdot$, max$\cdot$ and $\exists^{\mathrm{p}}\cdot$ and begin with an easy observation. Later, in Corollary 3.22 and Theorem 3.25, we will see that (under some assumption) the following two chains of embeddings can in general not be incorporated into a single chain (for example not for $\mathcal{X} = \mathrm{P}$).

Proposition 3.10. For any class of decision problems $\mathcal{X}$ closed under $\leq^{\mathrm{p}}_{\mathrm{m}}$ it holds that

$$\mathcal{X} \subseteq_{\equiv^{\mathrm{p}}_{\mathrm{T}}} \max\cdot \mathcal{X} \subseteq_{\equiv^{\mathrm{p}}_{\mathrm{T}}} \mathrm{wit}\cdot \mathcal{X} \quad \text{and} \quad \max\cdot \mathcal{X} \subseteq_{\equiv^{\mathrm{p}}_{\mathrm{T}}} \exists^{\mathrm{p}}\cdot \mathcal{X}$$

Proof. For any $A \in \mathcal{X}$ and a suitable polynomial p it holds that $f = \max_p\cdot\{\langle x, 1\rangle \mid x \in A\} \in \max\cdot \mathcal{X}$ and $f \equiv^{\mathrm{p}}_{\mathrm{T}} A$. For the second embedding let $A \in \mathcal{X}$ and p be a polynomial and consider the function $\max_p\cdot A$. Define $B = \{\langle\langle x, c\rangle, y\rangle \mid \langle x, y\rangle \in A, 1 \leq c \leq y < 2^{p(|x|)}\}$ and observe that $B \leq^{\mathrm{p}}_{\mathrm{m}} A$ and thus $B \in \mathcal{X}$. For a large enough polynomial q we have $\mathrm{wit}_q\cdot B \equiv^{\mathrm{p}}_{\mathrm{T}} \exists_q\cdot B \equiv^{\mathrm{p}}_{\mathrm{T}} \max_p\cdot A$ and thus the assertion is proved. □

Proposition 3.11. $\max\cdot \mathrm{NP} =_{\equiv^{\mathrm{p}}_{\mathrm{T}}} \mathrm{NP}$

Proof. For $\max\cdot \mathrm{NP} \subseteq_{\equiv^{\mathrm{p}}_{\mathrm{T}}} \mathrm{NP}$, observe that for any $f \in \max\cdot \mathrm{NP}$ we have $f \equiv^{\mathrm{p}}_{\mathrm{T}} \{\langle x, y\rangle \mid f(x) \geq y\} \in \mathrm{NP}$. The second part holds by Proposition 3.10. □

Proposition 3.12. For all $A \in \mathrm{NP}$, there is some $f \in \max\cdot \mathrm{P}$ such that $A \leq^{\mathrm{p}}_{\mathrm{T}} f$.

Proof. Let $A \in \mathrm{NP}$ and let $A = \exists_p\cdot R$ for some polynomial p and $R \in \mathrm{P}$. We may assume that $\langle x, 0\rangle \notin R$ for any x. Observe that $A \leq^{\mathrm{p}}_{\mathrm{T}} \max_p\cdot R \in \max\cdot \mathrm{P}$. □

Corollary 3.13. If $\mathrm{NP} \neq \mathrm{coNP}$ then $\max\cdot \mathrm{P} \not\subseteq_{\equiv^{\mathrm{p}}_{\mathrm{T}}} \mathrm{NP} \cap \mathrm{coNP}$.

Proof. We show that $\mathrm{NP} = \mathrm{coNP}$ under the assumption that $\max\cdot \mathrm{P} \subseteq_{\equiv^{\mathrm{p}}_{\mathrm{T}}} \mathrm{NP} \cap \mathrm{coNP}$. Let $A \in \mathrm{NP}$. Using Proposition 3.12, we obtain some $f \in \max\cdot \mathrm{P}$ such that $A \leq^{\mathrm{p}}_{\mathrm{T}} f$. By the assumption, there is some B such that $A \leq^{\mathrm{p}}_{\mathrm{T}} f \leq^{\mathrm{p}}_{\mathrm{T}} B \in \mathrm{NP} \cap \mathrm{coNP}$. Since $\mathrm{NP} \cap \mathrm{coNP}$ is closed under $\leq^{\mathrm{p}}_{\mathrm{T}}$, we get $A \in \mathrm{NP} \cap \mathrm{coNP}$. □

Proposition 3.14. If $\mathrm{P} \neq \mathrm{NP}$ then $\max\cdot \mathrm{P} \not\subseteq_{\equiv^{\mathrm{p}}_{\mathrm{T}}} \mathrm{P}$.

Proof. We assume that $\max\cdot \mathrm{P} \subseteq_{\equiv^{\mathrm{p}}_{\mathrm{T}}} \mathrm{P}$ and show $\mathrm{P} = \mathrm{NP}$. So let $A \in \mathrm{NP}$. Then there is some $B \in \mathrm{P}$ and a polynomial p such that $A = \exists_p\cdot B$. We may assume that $\langle x, 0\rangle \notin B$ for any x. Define $f = \max_p\cdot B$ and observe that $A \leq^{\mathrm{p}}_{\mathrm{T}} f$. Since f is equivalent to some set in P, we get $A \in \mathrm{P}$. □

Hemaspaandra et al. [HHN+95] show that $\mathrm{NPSV_t} \subseteq \mathrm{FP}^{\mathrm{NP}\cap\mathrm{coNP}}$, i. e. every function in $\mathrm{NPSV_t}$ reduces to a set in $\mathrm{NP} \cap \mathrm{coNP}$. We extend their proof and show that both classes are equivalent. Furthermore, if we drop the requirement that the single-valued functions are total, we exactly arrive at the class NP.

Proposition 3.15. It holds that

1. $\mathrm{NP} =_{\equiv^{\mathrm{p}}_{\mathrm{T}}} \mathrm{NPSV}$ and
2. $\mathrm{NP} \cap \mathrm{coNP} =_{\equiv^{\mathrm{p}}_{\mathrm{T}}} \mathrm{NPSV_t} \subseteq_{\equiv^{\mathrm{p}}_{\mathrm{T}}} \mathrm{coNPMV_t}$.

Proof. 1. Let $f \in \mathrm{NPSV}$ and define $A = \{\langle x, y\rangle \mid \exists z \in f(x)\colon y \leq z\}$. We have A in NP since on input $\langle x, y\rangle$ we can nondeterministically guess z and a witness for $z \in f(x)$. Furthermore, $f \equiv^{\mathrm{p}}_{\mathrm{T}} A$. For the other direction, let $A \in \mathrm{NP}$ and observe that $f \in \mathrm{NPSV}$ defined via $f(x) = \{1 \mid x \in A\}$ is equivalent to A.

2. Observe that it suffices to show that $\mathrm{NP} \cap \mathrm{coNP} \subseteq_{\equiv^{\mathrm{p}}_{\mathrm{T}}} (\mathrm{NP} \cap \mathrm{coNP})\mathrm{SV_t}$ and $\mathrm{NPSV_t} \subseteq_{\equiv^{\mathrm{p}}_{\mathrm{T}}} \mathrm{NP} \cap \mathrm{coNP}$. For the first inclusion, let $A \in \mathrm{NP} \cap \mathrm{coNP}$ and define $f \in$

$(\mathrm{NP} \cap \mathrm{coNP})\mathrm{SV}_\mathrm{t}$ via $f(x) = \{1 \mid x \in A\} \cup \{0 \mid x \notin A\}$. Since $A \in \mathrm{NP} \cap \mathrm{coNP}$, we have $f \in (\mathrm{NP} \cap \mathrm{coNP})\mathrm{SV}$ and f is obviously total. Furthermore, A and f are equivalent. In order to show the second inclusion, let $f \in \mathrm{NPSV}_\mathrm{t}$ and define $A = \{\langle x, y\rangle \mid y \leq z \text{ for } f(x) = \{z\}\}$. On input $\langle x, y\rangle$, we can guess the value of f together with a witness, verify it and accept accordingly. Since f is total and the machine notices if the value or the witness is wrong, we have $A \in \mathrm{NP} \cap \mathrm{coNP}$. As in the first part, we get $f \equiv^\mathrm{p}_\mathrm{T} A$. □

Before we turn to more complicated assertions and proofs, we give two results concerning coNPMV. Fenner et al. [FHOS97, Theorem 5.9] showed the equivalence of the first two statements in the following theorem. Their proof can be slightly adapted to also include the third statement [FGH+99].

Theorem 3.16 ([FHOS97],[FGH+99])**.** The following statements are equivalent:

1. $\mathrm{PH} = \Delta^\mathrm{P}_2$
2. For any $f \in \mathrm{NP}(2)\mathrm{MV}$ there is some $g \in \mathrm{NPMV}$ such that $f \leq^\mathrm{p}_\mathrm{T} g$.
3. For any $f \in \mathrm{coNPMV}$ there is some $g \in \mathrm{NPMV}$ such that $f \leq^\mathrm{p}_\mathrm{T} g$.

Proof. 1. ⇒ 2.: Assume $\mathrm{PH} = \Delta^\mathrm{P}_2$ and let $f \in \mathrm{NP}(2)\mathrm{MV}$ such that (without loss of generality) $0 \notin \mathrm{range}(f)$. Then there are $A, B \in \mathrm{NP}$ such that $f = \mathrm{wit}_p{\cdot}(A - B)$ for some polynomial p. Let $h = \max_p{\cdot}(A - B)$ and observe that $f \leq^\mathrm{p}_\mathrm{T} h \leq^\mathrm{p}_\mathrm{T} X$ for some $X \in \mathrm{PH} = \Delta^\mathrm{P}_2 = \mathrm{P}^\mathrm{NP}$. Hence, we have $X \leq^\mathrm{p}_\mathrm{T} \textsc{Sat}$ and thus $f \leq^\mathrm{p}_\mathrm{T} \mathrm{c}_\textsc{Sat} \in \mathrm{NPMV}$.

2. ⇒ 3.: This holds since $\mathrm{coNPMV} \subseteq \mathrm{NP}(2)\mathrm{MV}$.

3. ⇒ 1.: Assume that for any $f \in \mathrm{coNPMV}$ there is some $g \in \mathrm{NPMV}$ such that $f \leq^\mathrm{p}_\mathrm{T} g$ and let $A \in \Sigma^\mathrm{P}_2$. Then there is some polynomial p and some $B \in \mathrm{coNP}$ such that $A = \exists_p{\cdot}\, B$. Define $f = \mathrm{wit}_p{\cdot}\, B \in \mathrm{coNPMV}$. By assumption, there is some $g \in \mathrm{NPMV}$ such that $f \leq^\mathrm{p}_\mathrm{T} g$. We now have the reduction chain $A \leq^\mathrm{p}_\mathrm{T} f \leq^\mathrm{p}_\mathrm{T} g \leq^\mathrm{p}_\mathrm{T} \text{f-}\textsc{Sat} \leq^\mathrm{p}_\mathrm{T} \textsc{Sat}$ and thus $A \in \Delta^\mathrm{P}_2$. □

Corollary 3.17. If $\mathrm{PH} \neq \Delta^\mathrm{P}_2$ then $\mathrm{coNPMV} \not\subseteq_{\equiv^\mathrm{p}_\mathrm{T}} \mathrm{NPMV}$.

Proposition 3.18. If $\mathrm{PH} \neq \Sigma^\mathrm{P}_2$ then $\mathrm{coNPMV} \not\subseteq_{\equiv^\mathrm{p}_\mathrm{T}} \mathrm{coNPMV}_\mathrm{t}$.

Proof. We show that $\mathrm{coNPMV} \subseteq_{\equiv^\mathrm{p}_\mathrm{T}} \mathrm{coNPMV}_\mathrm{t}$ implies $\Sigma^\mathrm{P}_2 = \Pi^\mathrm{P}_2$. For this, let $A \in \Sigma^\mathrm{P}_2$. Then there is some polynomial p and some $B \in \mathrm{coNP}$ such that $A = \exists_p{\cdot}\, B$. Define $f = \mathrm{wit}_p{\cdot}\, B \in \mathrm{coNPMV}$ and observe that $\overline{A} \leq^\mathrm{p}_\mathrm{T} f$. By assumption, there is some $g \in \mathrm{coNPMV}_\mathrm{t}$ such that $f \leq^\mathrm{p}_\mathrm{T} g$ and hence we have $\overline{A} \leq^\mathrm{p}_\mathrm{T} g$ by some machine M with runtime p. To see that $\overline{A} \in \Sigma^\mathrm{P}_2$, observe that the following is true for every x: It holds that $x \in \overline{A}$ if and only if there is a set of oracle queries $Q = \{q_1, \ldots, q_k\}$ and corresponding answers $A = \{a_1, \ldots, a_k\}$ such that for all i, we have $(q_i, a_i) \in \mathrm{graph}(g)$ and the simulation of M on x only asks queries from Q and accepts when each q_i is answered by a_i. Note that the length of the description of Q and of A can be bounded by a polynomial in the length of x and $\mathrm{graph}(g) \in \mathrm{coNP}$. This means that $\overline{A} \in \Sigma^\mathrm{P}_2$ and thus $A \in \Pi^\mathrm{P}_2$. □

Fenner et al. [FGH+99] noted that the relation between NPMV and coNPMV is somewhat different to the relation between NP and coNP and that coNPMV seems to be more powerful than NPMV. Indeed, the previous corollary shows that $\mathrm{coNPMV} \subseteq_{\equiv^\mathrm{p}_\mathrm{T}} \mathrm{NPMV}$ is unlikely. This might suggest that $\mathrm{NPMV} \subseteq_{\equiv^\mathrm{p}_\mathrm{T}} \mathrm{coNPMV}$ could be true. We leave as an open question whether $\mathrm{NPMV} \subseteq_{\equiv^\mathrm{p}_\mathrm{T}} \mathrm{coNPMV}$ has unlikely complexity-theoretic consequences or perhaps even holds.

3.3 Search Problems Inequivalent to Any Decision Problem

In this section we show that under reasonable assumptions about complexity classes, there are multivalued functions in PMV_t that cannot be equivalent to any set. In other words, there are total NP search problems that are Turing-inequivalent to any decision problem. Beame et al. [BCE+98] showed that this holds relative to a generic oracle. This supports the conjecture that the complexity of functions in PMV (resp., the complexity of multiobjective problems, as we will see later) is in general not expressible in terms of sets.

In order to prove this result, we investigate the consequences of a multivalued function being equivalent to a set. We show that a PMV-function with sufficiently sparse domain can only be equivalent to a set from UP∩coUP and a function in NPMV_t that is equivalent to a set always has a refinement in NPSV_t.

Note that a multivalued function f is equivalent to a set if and only if the set of partial functions that are refinements of f has a minimal element with respect to the partial order $\leq^\mathrm{p}_\mathrm{T}$. In other words, a multivalued function f is not equivalent to any set if and only if no partial function that is a refinement of f is reducible (and thus equivalent) to f. As functions in $\mathcal{C}\mathrm{SV}$ for any class $\mathcal{C}$ cannot have more than one refinement, they are always equivalent to a set.

Property 3.19. For any $f \in \mathcal{C}\mathrm{MV}$ and any $A \subseteq \mathbb{N}$ the following is equivalent:

1. $f \equiv^\mathrm{p}_\mathrm{T} A$
2. There is a single-valued refinement g of f such that for any refinement h of f it holds that $g \leq^\mathrm{p}_\mathrm{T} h$.

Proof. "1. ⇒ 2.": Assume $f \equiv^\mathrm{p}_\mathrm{T} A$, so it holds that $f \leq^\mathrm{p}_\mathrm{T} A \leq^\mathrm{p}_\mathrm{T} f$. The first reduction always computes a certain single-valued refinement g of f. Since the reduction machine for $A \leq^\mathrm{p}_\mathrm{T} f$ has to accept any refinement of f, we also have $g \leq^\mathrm{p}_\mathrm{T} h$ for any refinement h of f by transitivity.

"2. ⇒ 1.": Define the set $A = \{\langle x, y\rangle \mid y \leq g(x)\}$ and observe that $g \equiv^\mathrm{p}_\mathrm{T} A$. A multivalued function is obviously reducible to any of its refinements and since g is reducible to f, we get $f \equiv^\mathrm{p}_\mathrm{T} g \equiv^\mathrm{p}_\mathrm{T} A$. □

Theorem 3.20. Let $t, m\colon \mathbb{N} \to \mathbb{N}$ such that $t(i) = 2^{2^{2^i}}$ and $m(i) = 2^i$. Let $f \in \mathrm{PMV}$ such that $\mathrm{dom}(f) = \{t(i) + k \mid i \in \mathbb{N}, 0 \leq k < m(i)\}$. If $f \equiv^\mathrm{p}_\mathrm{T} A$ for some $A \subseteq \mathbb{N}$, then $A \in \mathrm{UP} \cap \mathrm{coUP}$.

Proof. We first give the proof idea: Since $f \leq^\mathrm{p}_\mathrm{T} A \leq^\mathrm{p}_\mathrm{T} f$, there is a partial function h that is a refinement of f such that $h \leq^\mathrm{p}_\mathrm{T} f$ via machine M. To accept A in UP, we guess and verify a value for f for each query reachable from the input. We further check that our guess g in fact produces values for h by verifying that $g(y) = M^g(y) = h(y)$ holds for every reachable y. Since these guesses are unique, we arrive at a UP-computation.

Now we get to the main proof. Since $f \leq^\mathrm{p}_\mathrm{T} A$, there is some partial function $g\colon \mathbb{N} \to \mathbb{N}$ that is a refinement of f such that $g \leq^\mathrm{p}_\mathrm{T} A$. Furthermore, since $A \leq^\mathrm{p}_\mathrm{T} f$, we have $g \leq^\mathrm{p}_\mathrm{T} f$ via some polynomial-time oracle Turing machine M_g and $A \leq^\mathrm{p}_\mathrm{T} g$ via some polynomial-time oracle Turing machine M_A. For $i \in \mathbb{N}$ let $S_i = \{t(j) + k \mid 0 \leq j \leq i, 0 \leq k < m(j)\}$.

We now show $A \in \mathrm{UP} \cap \mathrm{coUP}$ by appropriately guessing an oracle O and simulating the computation of M_A with oracle O.

On input x, let $i \in \mathbb{N}$ minimal, such that M_A cannot query $t(i+1)$. Start by guessing and verifying a partial function O that maps each query $q \in S_i$ to an answer $a \in f(q)$. The length of the description of O is polynomial in $|x|$, since for some c,

$$|O| \leq c \sum_{j=0}^{i} m(j)|t(j)+m(j)|^c \leq c \sum_{j=0}^{i} |2\,t(j)|^{c+1} \leq c \sum_{j=0}^{i} (2+2^{2^{2^j}})^{c+1}$$
$$\leq c \sum_{j=0}^{i} 2^{(c+2)2^{2^j}} \leq c \sum_{j=0}^{(c+2)2^{2^i}} 2^j \leq c \cdot 2^{1+(c+2)2^{2^i}},$$

which is polynomial in $2^{2^{2^i}}$ and thus in $|x|$.

Remember that $g \leq_{\mathrm{T}}^{\mathrm{p}} f$ via M_g. On input $t(i)+m(i)-1$ (or smaller), there is some $c \in \mathbb{N}$ such that the largest number M_g can query is at most

$$2^{|t(i)+m(i)-1|^c} \leq 2^{|2\,t(i)|^c} \leq 2^{|t(i)|^{2c}} \leq 2^{\left(2^{2^{2^i}}\right)^{2c}} \leq 2^{2^{2c\,2^{2^i}}}.$$

For large enough i it holds that

$$2^{2^{2c\,2^{2^i}}} < 2^{\left(2^{2^{2^i}}\right)^2} = 2^{2^{2^{2^{i+1}}}} = t(i+1).$$

By encoding oracle answers into the program, we can assume that M_g only queries the oracle for inputs with i large enough for the above inequality to hold. Therefore, for all $q \in S_i$, $M_g^O(q)$ behaves as if O was a refinement of f and thus computes $g(q)$. We check if $M_g^O(q) = O(q)$ holds for all $q \in S_i$ and reject if this is not the case.

Observe that exactly one paths remains, namely the path where $O = g$ on S_i. Since M_A on input x cannot query $t(i+1)$, we can use O to simulate the reduction $A \leq_{\mathrm{T}}^{\mathrm{p}} g$ in deterministic polynomial time. Since we can negate the return value of M_A, we have shown that $A \in \mathrm{UP} \cap \mathrm{coUP}$. □

Proposition 3.21. If $f \in \mathrm{NPMV_t}$ such that $f \equiv_{\mathrm{T}}^{\mathrm{p}} A$ for some $A \subseteq \mathbb{N}$, then f has a refinement in $\mathrm{NPSV_t}$.

Proof. Let N be a nondeterministic polynomial-time machine that computes f. The following nondeterministic polynomial-time machine N' computes a singlevalued refinement of f: On input x, the machine deterministically simulates the reduction $f \leq_{\mathrm{T}}^{\mathrm{p}} A \leq_{\mathrm{T}}^{\mathrm{p}} f$ such that queries q to the oracle f are replaced by the nondeterministic simulation of N on q (the simulation stops at unsuccessful paths of N on q). The simulations of N on q always have successful paths, since f is total. Hence N' computes a total function.

Note that for the reduction $A \leq_{\mathrm{T}}^{\mathrm{p}} f$ it does not matter which element from $f(q)$ is returned to a query q. So each query r to A is answered the same way on all computation paths. Therefore, all successful paths of N' compute the same value from $f(x)$. □

Corollary 3.22. There exists a function $f \in \mathrm{PMV_t}$ such that $f \not\equiv_{\mathrm{T}}^{\mathrm{p}} A$ for all $A \subseteq \mathbb{N}$ if $\mathrm{UEEE} \cap \mathrm{coUEEE} \neq \mathrm{NEEE} \cap \mathrm{coNEEE}$ or $\mathrm{PMV_t} \not\subseteq_{\mathrm{c}} \mathrm{NPSV_t}$.

Proof. We first show the conclusion under UEEE ∩ coUEEE ≠ NEEE ∩ coNEEE. Let $t(n) = 2^{2^{2^n}}$. Proposition 2.12.4 provides a $B \in (\mathrm{NP} \cap \mathrm{coNP}) - (\mathrm{UP} \cap \mathrm{coUP})$ such that $B \subseteq \{t(i) + k \mid i \in \mathbb{N}, 0 \leq k < 2^i\}$. Choose a polynomial p and $R, R' \in \mathrm{P}$ such that $B = \exists_p \cdot R$ and $\overline{B} = \exists_p \cdot R'$. Let $S = \{\langle t(i) + k, y\rangle \in R \cup R' \mid i \in \mathbb{N}, 0 \leq k < 2^i\}$ and note that $S \in \mathrm{P}$. Let $f' = \mathrm{wit}_p \cdot S$. Observe that $\mathrm{dom}(f') = \exists_p \cdot S = \{t(i) + k \mid i \in \mathbb{N}, 0 \leq k < 2^i\}$. By Theorem 3.20, if $f' \equiv^{\mathrm{p}}_{\mathrm{T}} A$ for some $A \subseteq \mathbb{N}$, then $A \in \mathrm{UP} \cap \mathrm{coUP}$ and hence $B \in \mathrm{UP} \cap \mathrm{coUP}$ (since $B \leq^{\mathrm{p}}_{\mathrm{T}} f' \leq^{\mathrm{p}}_{\mathrm{T}} A$). The latter is a contradiction, and therefore $f' \not\equiv^{\mathrm{p}}_{\mathrm{T}} A$ for all $A \subseteq \mathbb{N}$. Furthermore, since $\mathrm{dom}(f') \in \mathrm{P}$ we can easily construct a total function $f \in \mathrm{PMV}$ such that $f \equiv^{\mathrm{p}}_{\mathrm{T}} f'$.

Now assume that there is a function $f \in \mathrm{PMV}_{\mathrm{t}}$ that does not have a refinement in $\mathrm{NPSV}_{\mathrm{t}}$. As the assumption that $f \equiv^{\mathrm{p}}_{\mathrm{T}} A$ for some $A \subseteq \mathbb{N}$ contradicts Proposition 3.21, we have shown the assertion. □

3.4 Decision Problems Inequivalent to Search Problems

Beigel et al. [BBFG91] show that if $\mathrm{DTIME}(2^{O(2^n)}) \neq \mathrm{NTIME}(2^{O(2^n)})$, then NP contains a language A for which search does not reduce to decision. This means that there is no $f \in \mathrm{PMV}$ such that $A = \mathrm{dom}(f)$ and $f \leq^{\mathrm{p}}_{\mathrm{T}} A$. Below we extend their construction and show that if EE ≠ NEE, then there exist sets in NP that are not polynomial-time Turing equivalent to any function in PMV, i. e. to any NP search problem. Furthermore, with a similar technique we show that if NEE ≠ coNEE, then there are sets in NP that are not polynomial-time Turing equivalent to any function in $\mathrm{coNPMV}_{\mathrm{t}}$. Note that our formulation uses the double exponential time classes $\mathrm{EE} = \mathrm{DTIME}(2^{2^{O(n)}})$ and $\mathrm{NEE} = \mathrm{NTIME}(2^{2^{O(n)}})$, while Beigel et al. use the slightly different classes $\mathrm{DTIME}(2^{2^{O(1)+n}})$ and $\mathrm{NTIME}(2^{2^{O(1)+n}})$.

Lemma 3.23. Let $f \in \mathrm{PMV}$ and $A \subseteq \{2^{2^i} \mid i \in \mathbb{N}\}$ such that $A \equiv^{\mathrm{p}}_{\mathrm{T}} f$. Then $A \in \mathrm{P}$.

Proof. Assume there exists an $f \in \mathrm{PMV}$ such that $A \equiv^{\mathrm{p}}_{\mathrm{T}} f$. So $f \leq^{\mathrm{p}}_{\mathrm{T}} A$ via an oracle Turing machine M whose running time is bounded by some polynomial q. We now show that by a simulation of M we can show that f is polynomial-time solvable.

Assume the input x has length n. Then the machine M cannot ask queries longer than $q(n)$. In particular, it cannot query $2^{2^{\lceil \log q(n) \rceil}} \geq 2^{q(n)}$. We simulate M for all possible oracles A' that satisfy $A' \subseteq \{2^{2^i} \mid i < \lceil \log q(n) \rceil\}$. Since there are at most $2^{\lceil \log q(n) \rceil} \leq 2q(n)$ such sets, this can be done in polynomial time.

Because of the above considerations, $M^{A'}$ must behave exactly as M^A for one of these oracles. Since $\mathrm{graph}(f) \in \mathrm{P}$, we can verify the correctness of all computed values and filter out wrong values. Hence, we can compute a refinement of f in polynomial time and thus obtain $A \in \mathrm{P}$, which is a contradiction. □

Lemma 3.24. Let f be a total multivalued function and $A \subseteq \{2^{2^i} \mid i \in \mathbb{N}\}$ such that $A \equiv^{\mathrm{p}}_{\mathrm{T}} f$. Then $A \leq^{\mathrm{p}}_{\mathrm{dtt(ctt)}} \mathrm{graph}(f)$.

Proof. Let f be a total multivalued function and $A \subseteq \{2^{2^i} \mid i \in \mathbb{N}\}$ such that $A \equiv^{\mathrm{p}}_{\mathrm{T}} f$. So $A \leq^{\mathrm{p}}_{\mathrm{T}} f \leq^{\mathrm{p}}_{\mathrm{T}} A$ via two polynomial-time oracle Turing machines M_1 and M_2 with running times r and s. Let $p(n) = s(r(n))$. On input x the machine M_1 can only ask queries with

length at most $r(|x|)$ and on inputs of length up to $r(|x|)$, the machine M_2 cannot query values starting from $2^{s(r(|x|))} = 2^{p(|x|)}$. In particular, it cannot query $2^{2^{\lceil \log p(|x|) \rceil}} \geq 2^{p(|x|)}$. So for each x and $A' = \{2^{2^i} \in A \mid i < \lceil \log p(|x|) \rceil\}$ it holds that

$$M_1^{M_2^{A'}}(x) = M_1^{M_2^{A}}(x) = \mathrm{c}_A(x).$$

We now argue for the correctness of the following statement and then show that it constitutes a $\leq^{\mathrm{p}}_{\mathrm{dtt(ctt)}}$-reduction from A to G. Let $Q(M_1^O)(x)$ be the queries that are asked during the simulation of the machine M_1 on input x with oracle O. Then the following holds for each x:

$$\begin{aligned} x \in A \iff \exists A' \subseteq \{2^{2^i} \mid i \in \mathbb{N}, i < \lceil \log p(|x|) \rceil\} \colon\ & M_1^{M_2^{A'}}(x) = 1 \text{ and} \\ \forall q \in Q(M_1^{M_2^{A'}})(x) \colon\ & M_2^{A'}(q) \in f(q) \end{aligned} \tag{3.1}$$

Let $x \in A$ and let $A' = \{2^{2^i} \in A \mid i < \lceil \log p(|x|) \rceil\}$ as above. Then we have $M_1^{M_2^{A'}}(x) = 1$ and $M_2^{A'}(q) \in f(q)$ for all $q \in Q(M_1^{M_2^{A'}})(x)$ since $|q| \leq r(|x|)$. Now assume $x \notin A$ and let A' be arbitrary. If $M_2^{A'}(q) \in f(q)$ for all $q \in Q(M_1^{M_2^{A'}})(x)$, the machine $M_2^{A'}$ behaves as a refinement of f on all queries that are asked by M_1. Since M_1 computes the reduction $A \leq^{\mathrm{p}}_{\mathrm{T}} f$ for every refinement of f, it must reject x with oracle $M_2^{A'}$ and thus the right-hand side is false.

It remains to show that statement (3.1) is a $\leq^{\mathrm{p}}_{\mathrm{dtt(ctt)}}$-reduction from A to $\mathrm{graph}(f)$. Since there are at most $2^{\lceil \log p(|x|) \rceil} \leq 2p(|x|)$ sets $A' \subseteq \{2^{2^i} \mid i \in \mathbb{N}, i < \lceil \log p(|x|) \rceil\}$, the existential quantifier is polynomially bounded. Furthermore, for each such A' the simulation $M_1^{M_2^{A'}}(x)$ can be carried out in polynomial time. During this simulation, all queries $q \in Q(M_1^{M_2^{A'}})(x)$ and their answers $M_2^{A'}(q)$ can be recorded. For the reduction, we end up with a polynomially-sized disjunction of polynomially-sized conjunctions of queries of the type $M_2^{A'}(q) \in f(q)$ and thus have a $\leq^{\mathrm{p}}_{\mathrm{dtt(ctt)}}$-reduction to $\mathrm{graph}(f)$. □

The combination of these two lemmas with the construction of very sparse sets from Proposition 2.12 yields the following arguments against embeddings.

Theorem 3.25. It holds that $\mathrm{NP} \not\subseteq_{\equiv^{\mathrm{p}}_{\mathrm{T}}} \mathrm{PMV}$ if EE ≠ NEE or NP has P-bi-immune sets.

Proof. Both assumptions provide a set $B \in \mathrm{NP} - \mathrm{P}$ such that $B \subseteq \{2^{2^i} \mid i \in \mathbb{N}\}$ as follows: For the first part, we can use Proposition 2.12. For the second part, we choose a P-bi-immune $L \in \mathrm{NP}$ and let $B = L \cap \{2^{2^i} \mid i \in \mathbb{N}\}$. From the P-bi-immunity of L it follows that $B \notin \mathrm{P}$.

In both cases, we apply Lemma 3.23 and obtain that B cannot be equivalent to any function in $\mathrm{PMV_t}$. □

Theorem 3.26. It holds that $\mathrm{NP} \cap \mathrm{coNP} \not\subseteq_{\equiv^{\mathrm{p}}_{\mathrm{T}}} \mathrm{PMV}$ if EE ≠ NEE or $\mathrm{NP} \cap \mathrm{coNP}$ has P-bi-immune sets.

Proof. We apply Proposition 2.12 and Lemma 3.23 in the same way as in the proof for Theorem 3.25. □

Theorem 3.27. It holds that NP $\not\subseteq_{\equiv_{\mathrm{T}}^{\mathrm{P}}}$ coNPMV$_{\mathrm{t}}$ if NEE $\neq$ coNEE.

Proof. Proposition 2.12 provides a $B \in \mathrm{NP} - \mathrm{coNP}$ such that $B \subseteq \{2^{2^i} \mid i \in \mathbb{N}\}$. By Lemma 3.24, this set cannot be polynomial-time Turing-equivalent to any function in coNPMV$_{\mathrm{t}}$ since coNP is closed under $\leq_{\mathrm{dtt(ctt)}}^{\mathrm{P}}$ by Property 2.23. □

Part I

Multiobjective Optimization

For many practical optimization problems, one objective function does not suffice. For example, when a wireless network operator wants to place base stations, the number of reachable customers is by far not the only objective function. At the same time, the installation costs should be minimized, problems caused by interference should be avoided and also maintenance plays a role. If not all of these objective functions can be transformed into a single value like money, or the exact formula for this transformation is not clear, this leads to so-called multiobjective optimization. This area has its origins in the late 1980s and has become increasingly popular since then [EG02]. We want to give a thorough structural and complexity-theoretic treatment of this subject, which has not been done so far. For a general overview of multiobjective optimization we refer to the survey by Ehrgott and Gandibleux [EG00] and the textbook by Ehrgott [Ehr05].

We extend the usual definition of single-objective (NP) optimization problems to multiple objectives in a straightforward way: For every instance there is a set of feasible solutions, each of which can be verified in polynomial time and furthermore, each of these solutions is assigned a multidimensional value. Since there are often multiple solutions for a single instance whose values are incomparable, there is in general no single optimal solution. This leads to the notion of the *Pareto set*, the set of all solutions for which there is no solution that is at least as good in all objectives and better in at least one objective (the filled circles in Figure 4.1 on page 62). Note that without any further information about the problem, each solution in the Pareto set must be considered to be equally good. So an algorithm that wants to solve a multiobjective problem should not pick some of the solutions from the Pareto set according to an internal heuristic, but rather give the user an impression of the whole Pareto set and allow a decision maker to pick one solution using further information or experience.

Unfortunately, the Pareto set is often exponentially large, so simply computing the whole Pareto set is not an option. For specific problems, different strategies were developed

to cope with this situation and most of the time, the problem is turned into an ordinary single-objective problem that is solved multiple times to obtain an impression of the Pareto set: One can restrict all but one objective to some (variable) minimal quality, apply a weighted sum or simply require that some arbitrary optimal solution should be generated. We will formalize these approaches as *solution notions* in chapter 5 and observe that they can differ drastically with respect to their complexity (unless P = NP), even for natural problems. In contrast, for single-objective problems, they are all equivalent (with respect to polynomial-time Turing reductions). The multivalued functions from the previous chapter will turn out helpful in formalizing the solution notions, as they capture one phenomenon that is crucial in multiobjective optimization: Since there is no single optimal solution, many algorithms solving a problem have some degree of freedom in the sense that they can generate one out of many valid outputs. Furthermore, the reduction we defined for multivalued functions (cf. Definition 3.2) is also suitable here: If we query an algorithm that solves a multiobjective problem, we cannot make any assumptions about the specific optimal or valid solution we get.

We will also investigate which complexities can appear for the different solution notions for a fixed problem and it will turn out that almost any combination of degrees in NP is possible. These observations exemplify the importance of clearly specifying the solution notion when stating that a multiobjective problem is NP-hard. Finally, we determine where the classes of multivalued functions defined by solution notions lie compared to known classes like NPMV or coNPMV. When these classes are compared to each other, for every combination we either show that the first class can be embedded in the second class or give evidence against such an embedding (via the results from chapter 3).

We already noted that for typical multiobjective problems, the number of Pareto optimal solutions is exponentially large. This is the case, for instance, for the traveling salesperson problem. This means that no polynomial-time algorithm can generate all optimal solutions. Papadimitriou and Yannakakis [PY00] have observed that there is always a polynomially-sized approximation of the Pareto set of arbitrarily good quality. The question remains if these approximations can be found by an algorithm in polynomial time. For many of the solution notions from chapter 5, one can define a corresponding approximate solution notion. We analyze these notions in chapter 6, compare them to each other and show how to transfer approximate solutions between these notions. In this section we will also see that some approximations for single-objective problems directly infer approximations for their multiobjective variants.

After this foundational analysis, we will turn to the traveling salesperson problem as one example of a multiobjective optimization problem and explicitly give approximation algorithms for some of its variants in chapter 7: For 2-TSP, the two-objective traveling salesperson problem (with triangle inequality), we give a deterministic 2-approximation, a randomized $(3/2 + \varepsilon, 2)$-approximation and a randomized $(3/2, 2 + \varepsilon)$-approximation for every $\varepsilon > 0$. We give evidence that, interestingly, there could be a trade-off in the approximability of the two objectives, i. e. in the approximation factors themselves. We also argue that it is difficult to substantially improve the given approximation ratios.

Another problem related to the usual traveling salesperson problem is its multiobjective maximization variant: Here, we show a randomized approximation ratio of $1/2$ and $2/3$, respectively. In contrast to earlier approximation algorithms for these problems, the algorithms that achieve these ratios are rather simple and apply a general strategy that

can be used to solve other multiobjective optimization problems. One key ingredient for this strategy is the so-called Beck-Fiala theorem [BF81]. It enables us to find a subset of a set of vectors such that the sum of this subset is roughly half of the sum of the whole set in every component.

We conclude our analysis of multiobjective optimization by proving a variant of the Beck-Fiala theorem in chapter 8. We show that we can find a subset of the vectors that has the same deviation from the exact half as in the Beck-Fiala theorem, only that the structure of the subset is more convenient: If we go through the list of vectors, the number of switches between excluding and including vectors is constant in the number of vectors. In other words, the set of indices that make up the chosen subset is a constant number of unions of (integer) intervals. To achieve this combinatorial result we use analytical and topological methods.

Chapter 4

Definitions

Most of the literature on multiobjective optimization either tries to find approximation algorithms for specific problems or investigates these problems from a purely mathematical point of view, mostly disregarding computability and complexity aspects. The study of the general complexity of multiobjective optimization (especially approximation) was initiated by Papadimitriou and Yannakakis [PY00] and also Ehrgott [Ehr00] obtained some general results. We continue their work, try to find precise and general definitions for multiobjective optimization problems and their solution notions and, building upon that, perform a structural and extensive study in this and the following chapters.

4.1 Basic Definitions and Properties

We start with basic definitions and terminology in the context of multiobjective optimization. Examples of multiobjective optimization problems can be found in section 4.2. We first formally define multiobjective problems and related notation. Then we define different search and value notions. Search notions are multivalued functions that output solutions while value notions output only the values of the solutions. We conclude this section by observing that for a single-objective problem, all search notions are equivalent and all value notions are equivalent and we give general upper bounds for the solution notions.

Definition 4.1. Let $k \geq 1$. A *k-objective* NP *optimization problem* (*k-objective problem*, for short) is a triple $(S, f, \leftarrow)$ where S specifies the *set of feasible solutions* for an input, f is the (multidimensional) *objective function* and $\leftarrow$ defines the direction of optimization. More specifically:

- $S\colon \mathbb{N} \to 2^{\mathbb{N}}$ maps an instance $x \in \mathbb{N}$ to the set of feasible solutions for this instance, denoted as $S^x = S(x) \subseteq \mathbb{N}$. There must be some polynomial p such that for every $x \in \mathbb{N}$ and every $s \in S^x$ it holds that $s \leq 2^{p(|x|)}$ and the set $\{\langle x, s\rangle \mid x \in \mathbb{N}, s \in S^x\}$ must be polynomial-time decidable. All these conditions on S can be equivalently expressed by $S \in \mathrm{PMV}$.
- $f\colon \{\langle x, s\rangle \mid x \in \mathbb{N}, s \in S^x\} \to \mathbb{N}^k$ maps an instance $x \in \mathbb{N}$ and a solution $s \in S^x$ to its value, denoted by $f^x(s) \in \mathbb{N}^k$. The function f must be polynomial-time computable.

- $\leftarrow \subseteq \mathbb{N}^k \times \mathbb{N}^k$ is a partial order on the values of solutions. It must hold that $(a_1, \ldots, a_k) \leftarrow (b_1, \ldots, b_k) \iff a_1 \leftarrow_1 b_1 \wedge \cdots \wedge a_k \leftarrow_k b_k$, where $\leftarrow_i$ is $\leq$ if the i-th objective is minimized, and $\leftarrow_i$ is $\geq$ if the i-th objective is maximized.

We also use $\leq$ as the partial order $\leftarrow$ where $\leftarrow_i = \leq$ for all i and $\geq$ is used analogously. The projection of f^x to the i-th component is denoted as f_i^x where $f_i^x(s) = v_i$ if $f^x(s) = (v_1, \ldots, v_k)$. Initially, the relation $\leftarrow$ is only used on values $a, b \in \mathbb{N}^k$. If $a \leftarrow b$ we say that *a weakly dominates b* (i. e., *a* is at least as good as *b*). If $a \leftarrow b$ and $a \neq b$ we say that *a dominates b*. Note that $\leftarrow$ always points in the direction of the better value. If f and x are clear from the context, then we extend $\leftarrow$ to combinations of values and solutions. So we can talk about weak dominance between solutions, and we write $s \leftarrow t$ if $f^x(s) \leftarrow f^x(t)$, $s \leftarrow c$ if $f^x(s) \leftarrow c$, and so on, where $s, t \in S^x$ and $c \in \mathbb{N}^k$. Furthermore, we define $\mathrm{opt}_\leftarrow \colon 2^{\mathbb{N}^k} \to 2^{\mathbb{N}^k}$, $\mathrm{opt}_\leftarrow(M) = \{y \in M \mid \forall z \in M[z \leftarrow y \Rightarrow z = y]\}$ as a function that maps sets of values to sets of optimal values. The operator $\mathrm{opt}_\leftarrow$ is also applied to sets of solutions $S' \subseteq S^x$ as $\mathrm{opt}_\leftarrow(S') = \{s \in S' \mid f^x(s) \in \mathrm{opt}_\leftarrow(f^x(S'))\}$. If even $\leftarrow$ is clear from the context, we write $S^x_{\mathrm{opt}} = \mathrm{opt}_\leftarrow(S^x)$ and $\mathrm{opt}_i(S') = \{s \in S' \mid f_i^x(s) \in \mathrm{opt}_{\leftarrow_i}(f_i^x(S'))\}$. The set S^x_{opt} is called the *Pareto set* (the set of *(Pareto-)optimal solutions*) for the instance x and the set of values of these solutions, $f(S^x)_{\mathrm{opt}}$ is called the *Pareto curve*, though we do not always distinguish these two concepts clearly.

Definition 4.2. For every k-objective problem $\mathcal{O} = (S, f, \leftarrow)$ and all $1 \leq i \leq k$ we define the *search notions arbitrary optimum* (A-$\mathcal{O}$), *dominating solution* (D-$\mathcal{O}$), *specific optimum* (S-$\mathcal{O}$), *constraint optimum* ($\mathrm{C_i}$-$\mathcal{O}$), *lexicographic optimum* (L-$\mathcal{O}$), and *weighted sum optimum* (W-$\mathcal{O}$) as multivalued functions from $\mathbb{N}$ to $\mathbb{N}$, where

$$\begin{aligned}
\text{A-}\mathcal{O}(x) &= S^x_{\mathrm{opt}} \\
\text{D-}\mathcal{O}(\langle x, \langle c \rangle\rangle) &= \{y \in S^x \mid y \leftarrow c\} \\
\text{S-}\mathcal{O}(\langle x, \langle c \rangle\rangle) &= \left\{y \in S^x_{\mathrm{opt}} \mid y \leftarrow c\right\} \\
\mathrm{C_i}\text{-}\mathcal{O}(\langle x, \langle c \rangle\rangle) &= \mathrm{opt}_i\left(\left\{s \in S^x \mid f_j^x(s) \leftarrow_j c_j \text{ for all } j \neq i\right\}\right) \\
\text{L-}\mathcal{O}(x) &= \mathrm{opt}_k(\ldots(\mathrm{opt}_2(\mathrm{opt}_1(S^x)))\ldots) \\
\text{W-}\mathcal{O}(\langle x, \langle \omega \rangle\rangle) &= \left\{y \in S^x \mid \forall s \in S^x \left[\omega^{\mathrm{T}} f^x(y) \leftarrow_1 \omega^{\mathrm{T}} f^x(s)\right]\right\}
\end{aligned}$$

for all $x \in \mathbb{N}$ and $c, \omega \in \mathbb{N}^k$, where $\omega^{\mathrm{T}} f^x(y) = \sum_{j=1}^k \omega_j f_j^x(y)$ is the inner product of the vectors ω and $f^x(y)$. For the weighted sum optimum notion, we assume that all objectives are to be maximized or all objectives are to be minimized.

Before discussing these notions, we note that each search notion maps to sets of solutions, which, in turn, map to values in $\mathbb{N}^k$ via f. Hence, each search notion naturally motivates a *value notion* for the problem that is defined as follows.

Definition 4.3. For every k-objective problem $\mathcal{O} = (S, f, \leftarrow)$ we define the *value notion* $\mathrm{Val}(\mathcal{X}\text{-}\mathcal{O})$ (of the search notion $\mathcal{X}$-$\mathcal{O}$) as a multivalued function from $\mathbb{N}$ to $\mathbb{N}^k$, where

$$\mathrm{Val}(\mathcal{X}\text{-}\mathcal{O})(\varphi) = f^x(\mathcal{X}\text{-}\mathcal{O}(\varphi))$$

for all $\varphi \in \mathbb{N}$ and $\mathcal{X} \in \{\mathrm{A}, \mathrm{D}, \mathrm{S}, \mathrm{C}_1, \mathrm{C}_2, \ldots, \mathrm{C_k}, \mathrm{L}, \mathrm{W}\}$, where x is the problem instance encoded in φ, and $\mathcal{X} = W$ only if all objectives are to be maximized (resp., minimized).

For a graphical illustration of the solution notions, please confer Figure 4.1. Note that since A-$\mathcal{O}$ is defined as a multivalued function, the task is not to output all optimal solutions, i. e. the whole set S^x_{opt}, but rather to choose one arbitrary element from this set and output it. Hence A-$\mathcal{O}$ is polynomial-time solvable if for all input instances $x \in \mathbb{N}$ we can decide if $S^x \neq \emptyset$ and further find some arbitrary optimal solution $s \in S^x_{\text{opt}}$ in polynomial time. Analogously, the specific optimum notion searches for optimal solutions that are restricted to be at least as good as the constraint vector $c \in \mathbb{N}^k$, whereas the dominating solution notion does not require the solutions to be optimal. For a single-objective problem, the dominating solution notion corresponds to a formulation of a problem where the optimum can be found by binary search. The constraint optimum notion for the i-th objective searches solutions that are at least as good as c for all objectives $j \neq i$ and optimal for objective i, while the lexicographical optimum notion searches for solutions that are optimal according to some fixed order of objectives (here: $1, 2, \ldots, k$). Finally, the weighted sum optimum notion searches for solutions such that the sum of all objectives weighted with the weight vector $\omega \in \mathbb{N}^k$ is optimal, i. e. it aggregates several objective functions into one. Note that $\leftarrow_1$ is used in the definition, but can be equivalently replaced by any $\leftarrow_i$, since they are all equal. In the literature, C-$\mathcal{O}$ is also known as *bottleneck optimization* and L-$\mathcal{O}$ as *hierarchical optimization*. Moreover, W-$\mathcal{O}$ is a particular *normalization approach*, since a norm is used to aggregate several cost functions into one. We will get to general normalized approaches in section 6.3.

From Figure 4.1, one can already deduce that W-$\mathcal{O}$ cannot be forced to output some of the Pareto-optimal solutions, since the Pareto-curve is not convex. This suggests that weighted sums, which are actually quite popular in practice, are not the right way to attack multiobjective problems or at least will not give a complete impression of the Pareto set. Nevertheless, since multiobjective problems are not modeled by arbitrary functions but obey some complexity-theoretic restrictions, it is not a priori clear that W-$\mathcal{O}$ is "weaker" (with respect to reductions) than for example S-$\mathcal{O}$. It could well be that information about the solutions that are not reachable via W-$\mathcal{O}$ is encoded in solutions that are reachable. In fact, we will show that W-$\mathcal{O}$ and S-$\mathcal{O}$ are somewhat incomparable with respect to their complexity (cf. Figure 5.1 on page 69 and Table 5.1 on page 72).

Note that for $k = 1$, the above definitions correspond to the notion of a (single-objective) NP optimization problem as usually defined, for example in the book by Ausiello et. al. [APMS$^+$99]. Furthermore, for a single-objective NP optimization problem $\mathcal{O}$, one normally only considers the solution notion A-$\mathcal{O}$. The following proposition explains why the distinction into different solution notions is usually not made for a single objective: All notions defined above are equivalent in this case. This mainly stems from the fact that the (value of the) optimal solution is unique.

Proposition 4.4. For any single-objective problem $\mathcal{O}$ all search notions are equivalent and all value notions are equivalent, i. e.

$$\text{A-}\mathcal{O} = \text{L-}\mathcal{O} \equiv^{\text{p}}_{\text{T}} \text{W-}\mathcal{O} \equiv^{\text{p}}_{\text{T}} \text{C}_1\text{-}\mathcal{O} \equiv^{\text{p}}_{\text{T}} \text{D-}\mathcal{O} \equiv^{\text{p}}_{\text{T}} \text{S-}\mathcal{O}$$

and

$$\text{Val(A-}\mathcal{O}) = \text{Val(L-}\mathcal{O}) \equiv^{\text{p}}_{\text{T}} \text{Val(W-}\mathcal{O}) \equiv^{\text{p}}_{\text{T}} \text{Val(C}_1\text{-}\mathcal{O}) \equiv^{\text{p}}_{\text{T}} \text{Val(D-}\mathcal{O}) \equiv^{\text{p}}_{\text{T}} \text{Val(S-}\mathcal{O}).$$

Proof. We begin with the search notions. The first equality is clear from the definition. L-$\mathcal{O} \leq^{\text{p}}_{\text{T}}$ W-$\mathcal{O}$ is achieved by a single query with weight 1. The reduction W-$\mathcal{O} \leq^{\text{p}}_{\text{T}}$ C$_1$-$\mathcal{O}$

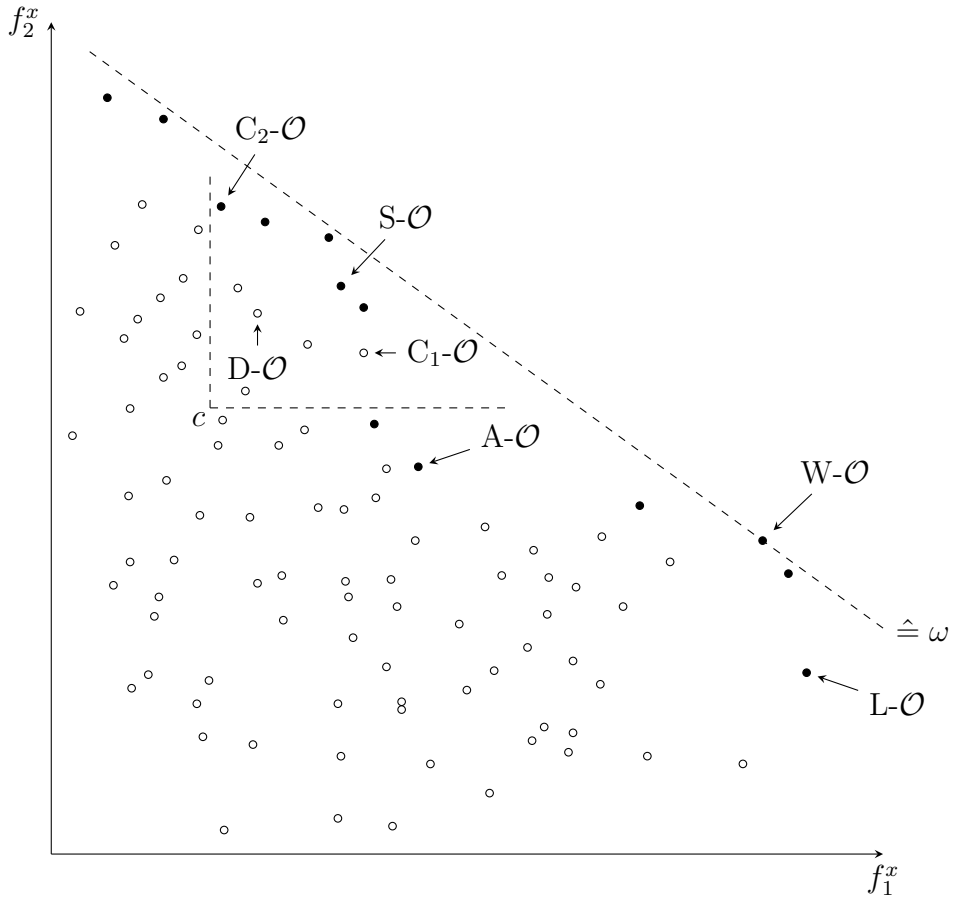


Figure 4.1: Illustration of the solution notions for a single instance x of a two-objective maximization problem $\mathcal{O} = (S, f, \geq)$. Circles denote the values of feasible solutions while filled circles represent Pareto-optimal values (i. e. the Pareto curve). For each solution notion, (the value of) one possible correct output is marked by an arrow, where the coordinates of the point c define the constraints for C$_i$-$\mathcal{O}$, S-$\mathcal{O}$ and D-$\mathcal{O}$ and the input ω for W-$\mathcal{O}$ is given indirectly by the inclination of the diagonal dashed line.
Note that in this situation, the second valid output for C$_1$-$\mathcal{O}$ is the (solution represented by the) point directly above the marked point, while for C$_2$-$\mathcal{O}$ there is exactly one valid output (assuming that there are no two solutions with the same value). The notions A-$\mathcal{O}$, S-$\mathcal{O}$ and D-$\mathcal{O}$ can also output different solutions while W-$\mathcal{O}$ and L-$\mathcal{O}$ have no other option. Observe that W-$\mathcal{O}$ cannot be forced to output the solution labeled by A-$\mathcal{O}$ for any weight vector.

holds by a single query with constraint vector (1) (it is ignored). Note that W-$\mathcal{O}$ is allowed to output any solution if the weight is zero. The next reduction, C_1-$\mathcal{O} \leq_{\mathrm{T}}^{\mathrm{p}}$ D-$\mathcal{O}$ is the only one that requires more than one query: We have to make a binary search for the optimal value. D-$\mathcal{O} \leq_{\mathrm{T}}^{\mathrm{p}}$ S-$\mathcal{O}$ is clear from the definition and S-$\mathcal{O} \leq_{\mathrm{T}}^{\mathrm{p}}$ L-$\mathcal{O}$ is done by a single query to L-$\mathcal{O}$ followed by a check if the value of the returned solution (if any) satisfies the constraint.

Since the above reductions only used the values of the returned solutions, they are equally valid for the respective value notions. □

Here, the question for the circumstances under which even the value notions are equivalent to the search notions arises. This will be covered at the end of section 5.1, where we will see that this question is connected to the notion of search reduces to decision.

When we are only interested in exact solutions for multiobjective problems (in the lens of Turing equivalence), we can concentrate on problems where all objectives are to be maximized. In sections 6.3 and 6.4 we will see that this distinction is important for approximate solution notions.

Proposition 4.5. For every k-objective problem $\mathcal{O} = (S, f, \leftarrow)$ there is a k-objective problem $\mathcal{O}' = (S, f', \geq)$ such that for all $\mathcal{X} \in \{\mathrm{A}, \mathrm{D}, \mathrm{S}, \mathrm{C}_1, \mathrm{C}_2, \ldots, \mathrm{C}_\mathrm{k}, \mathrm{L}, \mathrm{W}\}$

$$\mathcal{X}\text{-}\mathcal{O} = \mathcal{X}\text{-}\mathcal{O}' \qquad \text{and} \qquad \mathrm{Val}(\mathcal{X}\text{-}\mathcal{O}) \equiv_{\mathrm{T}}^{\mathrm{p}} \mathrm{Val}(\mathcal{X}\text{-}\mathcal{O}')$$

(where $\mathcal{X} = W$ is only considered for $\leftarrow \in \{\leq, \geq\}$).

Proof. Since f must be polynomial-time computable, there is a polynomial p such that for every $i \in \{1, \ldots, k\}$, $f_i^x(s) \leq 2^{p(|x|)}$. For every i such that $\leftarrow_i = \leq$, let $f'^x_i(s) = 2^{p(|x|)} - f_i^x(s)$ and $f'^x_i(s) = f_i^x(s)$ for all other i. Observe that the assertions hold. □

If we look close enough, we see that the three notions W-$\mathcal{O}$, L-$\mathcal{O}$ and $\mathrm{C_i}$-$\mathcal{O}$ are actually single-objective problems in disguise. This can be roughly explained by the fact that the solutions that they are allowed to output always have the same value (or at least they can be forced to a unique value for $\mathrm{C_i}$-$\mathcal{O}$ and W-$\mathcal{O}$).

Proposition 4.6. For every k-objective problem $\mathcal{O} = (S, f, \leftarrow)$ there exist single-objective problems $\mathcal{O}_\mathrm{W}$ (if $\leftarrow \in \{\leq, \geq\}$), $\mathcal{O}_\mathrm{C}$ and $\mathcal{O}_\mathrm{L}$ such that for all $1 \leq i \leq k$,

$$\mathrm{W}\text{-}\mathcal{O} = \mathrm{A}\text{-}\mathcal{O}_\mathrm{W}, \quad \mathrm{C_i}\text{-}\mathcal{O} = \mathrm{A}\text{-}\mathcal{O}_\mathrm{C} \quad \text{and} \quad \mathrm{L}\text{-}\mathcal{O} = \mathrm{A}\text{-}\mathcal{O}_\mathrm{L}.$$

Proof. Because of Proposition 4.5, we can assume that $\leftarrow = \geq$.

For $\mathcal{O}_\mathrm{W}$ and $\mathcal{O}_\mathrm{C}$ this is immediate from the definition where the problems are defined as $\mathcal{O}_\mathrm{W} = (S_\mathrm{W}, f_\mathrm{W}, \geq)$ with $S_\mathrm{W}^{\langle x, \langle \omega_1, \ldots, \omega_k \rangle\rangle} = S^x$ and $f_\mathrm{W}^{\langle x, \langle \omega_1, \ldots, \omega_k \rangle\rangle}(s) = \sum_{i=1}^k \omega_i f_{\mathrm{W},i}^x(s)$, $\mathcal{O}_\mathrm{C} = (S_\mathrm{C}, f_\mathrm{C}, \geq)$ with $S_\mathrm{C}^{\langle x, \langle c \rangle\rangle} = \{s \in S^x \mid f_j^x(s) \geq c_j \text{ for all } j \neq i\}$ and $f_\mathrm{C}^{\langle x, \langle c \rangle\rangle}(s) = f_i(s)$.

For $\mathcal{O}_\mathrm{L}$ it is more complicated, but also straightforward: The new objective function is a weighted sum of the original objective functions. More specifically, assume that there is a polynomial p such that for every $i \in \{1, \ldots, k\}$, $f_i^x(s) < 2^{p(|x|)}$. Then define $\mathcal{O}_\mathrm{L} = (S, f_\mathrm{L}, \geq)$, where

$$f_\mathrm{L}^x(s) = \sum_{i=1}^k 2^{p(|x|)} f_i(s)$$

and observe that the assertion holds. □

As a last result of this section, we give upper bounds for the solution notions in terms of classes of multivalued functions they lie in.

Proposition 4.7. Let $\mathcal{O} = (S, f, \leftarrow)$ be a k-objective problem.

1. $\mathcal{X}$-$\mathcal{O} \in$ coNPMV for all $\mathcal{X} \in \{\mathrm{A}, \mathrm{S}, \mathrm{C}_1, \mathrm{C}_2, \ldots, \mathrm{C}_\mathrm{k}, \mathrm{L}, \mathrm{W}\}$
2. Val($\mathcal{X}$-$\mathcal{O}$) $\in$ NP(2)MV for all $\mathcal{X} \in \{\mathrm{A}, \mathrm{S}, \mathrm{C}_1, \mathrm{C}_2, \ldots, \mathrm{C}_\mathrm{k}, \mathrm{L}, \mathrm{W}\}$
3. D-$\mathcal{O} \in$ PMV and Val(D-$\mathcal{O}$) $\in$ NPMV

Proof. 1. Let $\mathcal{X} \in \{\mathrm{A}, \mathrm{S}, \mathrm{C}_1, \mathrm{C}_2, \ldots, \mathrm{C}_\mathrm{k}, \mathrm{L}, \mathrm{W}\}$. By the definition of multiobjective problems and search notions, $(x, y) \in \mathrm{graph}(\mathcal{X}\text{-}\mathcal{O})$ implies that $|y|$ is polynomially bounded in the length of x. Further observe that $\mathrm{graph}(\mathcal{X}\text{-}\mathcal{O}) \in$ coNP since the optimality (with respect to $\mathcal{X}$-$\mathcal{O}$) of a solution can be decided by checking that there is no solution that is "better" than the current solution. Hence we obtain $\mathcal{X}$-$\mathcal{O} \in$ coNPMV.

2. Let $\mathcal{X} \in \{\mathrm{A}, \mathrm{S}, \mathrm{C}_1, \mathrm{C}_2, \ldots, \mathrm{C}_\mathrm{k}, \mathrm{L}, \mathrm{W}\}$. Similarly as in the previous part, for $(x, y) \in \mathrm{graph}(\mathrm{Val}(\mathrm{A}\text{-}\mathcal{O}))$, the length of y is polynomially bounded in the length of x since f is computable in polynomial time. Further observe that $(x, y) \in \mathrm{graph}(\mathrm{Val}(\mathrm{A}\text{-}\mathcal{O}))$ if and only if

$$(\exists s \in S^x\colon\ f(s) = y) \wedge \neg(\exists s \in S^x\colon\ f(s) \neq y \wedge f(s) \leftarrow y),$$

which is an NP(2)-condition. Since similar conditions can be formulated for the other value notions, we obtain Val($\mathcal{X}$-$\mathcal{O}$) $\in$ NP(2)MV.

3. Again, solutions and values are polynomially bounded. Further observe that $\mathrm{graph}(\mathrm{D}\text{-}\mathcal{O}) \in \mathrm{P}$ and $\mathrm{graph}(\mathrm{Val}(\mathrm{D}\text{-}\mathcal{O})) \in \mathrm{NP}$, because $(\langle x, c\rangle, s) \in \mathrm{graph}(\mathrm{D}\text{-}\mathcal{O}) \iff s \in S^x$ and $y \leftarrow c$, which can be tested in polynomial time, whereas $(\langle x, c\rangle, y) \in \mathrm{graph}(\mathrm{Val}(\mathrm{D}\text{-}\mathcal{O}))$ needs to further check if a solution $s \in S^x$ with $f^x(s) = y$ exists. □

4.2 Examples of Multiobjective Problems

To illustrate the formal definition in the previous section, we want to give some examples of multiobjective problems. Furthermore, algorithms for some of these problems will be used in chapter 7 to help solving multiobjective traveling salesperson problems. All of these example problems except for the first two are so-called *linear* problems (cf. Definition 5.2). The preference of linear problems in (the theory of) multiobjective optimization stems from the fact that they are easy to describe and can be handled theoretically. In contrast, practical problems are often non-linear but cannot be succinctly defined.

Definition 4.8 (Minimimum Lateness and Weighted Flowtime Scheduling)**.**

2-LWF $= (S, f, \leq)$ where

- instances are triples $\langle P, D, W\rangle$ such that
  - $P = (p_1, \ldots, p_n) \in \mathbb{N}^n$ are processing times
  - $D = (d_1, \ldots, d_n) \in \mathbb{N}^n$ are due dates
  - $W = (w_1, \ldots, w_n) \in \mathbb{N}^n$ are weights,
- $S^{\langle P,D,W\rangle} = \{\pi \mid \pi \text{ is a permutation of } \{1, \ldots, n\}\}$ and

- $f^{\langle P,D,W\rangle}(\pi) = (L_{\max}, \sum_{j=1}^{n} w_j C_j)$ where
  - the *completion time* of job j is $C_j = p_{\pi(1)} + p_{\pi(2)} + \cdots + p_{\pi(j)}$.
  - the *maximum lateness* is $L_{\max} = \max\{C_j - d_j \mid 1 \leq j \leq n\}$.
  - the *weighted flowtime* is $\sum_{j=1}^{n} w_j C_j$.

Furthermore, the problem 2-LF is the variant of 2-LWF where all weights are 1, i. e., $W = (1, \ldots, 1)$.

Note that 2-LWF and 2-LF do not strictly conform to the definition of multiobjective optimization problems because f can have negative values. Nonetheless, since f is polynomial-time computable, one can easily construct an equivalent problem where the solutions only have non-negative values by adding an appropriate number depending only on the length of the input.

Definition 4.9 (Minimum Quadratic Diophantine Equation)**.**

2-QDE $= (S, f, \leq)$ where

- instances are encoded triples $\langle a, b, c\rangle$ of natural numbers,
- $S^{\langle a,b,c\rangle} = \{\langle x, y\rangle \mid ax^2 + by^2 - c \geq 0\}$, and
- $f^{\langle a,b,c\rangle}(\langle x, y\rangle) = (x^2, y^2)$.

Definition 4.10 (k-Objective Maximum Weighted Satisfiability)**.**

k-WSAT $= (S, f, \geq)$ where

- instances are sets of clauses $C = \{C_1, \ldots, C_r\}$ over m variables with labels $l\colon \{C_1, \ldots, C_r\} \to \mathbb{N}^k$,
- $S^{\langle C,l\rangle} = \{I \mid I\colon \{x_1, \ldots, x_m\} \to \{0, 1\}\}$ and
- $f^{\langle C,l\rangle}(I) = \sum\{l(C_i) \mid I(C_i) = 1\}$.

Definition 4.11 (k-Objective Maximum (Weight) Clique)**.**

k-MaxClique $= (S, f, \geq)$ where

- instances are $\mathbb{N}^k$-vertex-labeled undirected graphs $G = (V, E, l)$,
- $S^{\langle G\rangle} = \{C \subseteq V \mid \forall x, y \in C, x \neq y\colon \{x, y\} \subseteq E\}$ and
- $f^{\langle G\rangle}(C) = \sum_{v\in C} l(v)$.

Definition 4.12 (k-Objective Shortest Path)**.**

k-SP $= (S, f, \leq)$ where

- instances are $\mathbb{N}^k$-edge-labeled directed graphs $G = (V, E, l)$ and two distinct vertices $s, t \in V$,
- $S^{\langle G,s,t\rangle} = \{P \subseteq E \mid P$ is a path from s to t in $G\}$ and
- $f^{\langle G,s,t\rangle}(P) = \sum_{e\in P} l(e)$.

Definition 4.13 (k-Objective Minimum (Weight) Perfect Matching)**.**

k-MM $= (S, f, \leq)$ where

- instances are $\mathbb{N}^k$-edge-labeled undirected graphs $G = (V, E, l)$,
- $S^{\langle G \rangle} = \{M \subseteq E \mid M \text{ is a perfect matching in } G\}$ and
- $f^{\langle G \rangle}(P) = \sum_{e \in M} l(e)$.

Definition 4.14 (k-Objective Minimum (Weight) Spanning Tree)**.**

k-MST $= (S, f, \leq)$ where

- instances are $\mathbb{N}^k$-edge-labeled undirected graphs $G = (V, E, l)$,
- $S^{\langle G \rangle} = \{T \subseteq E \mid T \text{ is a spanning tree of } G\}$ and
- $f^{\langle G \rangle}(T) = \sum_{e \in T} l(e)$.

Definition 4.15 (k-Objective Maximum Directed Cycle Cover)**.**

k-MaxDCC $= (S, f, \geq)$ where

- instances are $\mathbb{N}^k$-edge-labeled directed complete graphs $G = (V, E, l)$,
- $S^{\langle G \rangle} = \{C \subseteq E \mid C \text{ is a cycle cover of } G\}$ and
- $f^{\langle G \rangle}(C) = \sum_{e \in C} l(e)$.

The restriction of k-MaxDCC to undirected graphs is called k-MaxUCC (note that cycles have length at least three in that case).

Definition 4.16 (k-Objective Traveling Salesperson)**.**

k-TSP $= (S, f, \leq)$ where

- instances are $\mathbb{N}^k$-edge-labeled directed multigraphs $G = (V, E, l)$,
- $S^{\langle G \rangle} = \{W \subseteq E \mid W \text{ is a closed spanning walk of } G\}$ and
- $f^{\langle G \rangle}(W) = \sum_{e \in W} l(e)$.

We define the multiobjective traveling salesperson problem on multigraphs and with multiple visits as this best generalizes the single-objective problem on metric graphs to multiple objectives. For a discussion we refer to chapter 7.

Definition 4.17 (k-Objective Maximum Traveling Salesperson)**.**

k-MaxATSP $= (S, f, \geq)$ where

- instances are $\mathbb{N}^k$-edge-labeled directed complete graphs $G = (V, E, l)$,
- $S^{\langle G \rangle} = \{C \subseteq E \mid C \text{ is a Hamiltonian cycle of } G\}$ and
- $f^{\langle G \rangle}(C) = \sum_{e \in C} l(e)$.

The letter A in k-MaxATSP stands for "asymmetric", although the labeling function is not required to be asymmetric. The restriction of k-MaxATSP to undirected graphs (or symmetric edge labels) is called k-MaxSTSP.

Chapter 5

Structural Properties of Solution Notions

Having formally defined the mathematical object of a multiobjective optimization problem and its solution notions we now analyze their complexity-theoretic properties in general. For this, we take two approaches: First, we analyze the reducibilities between solution notions for the same fixed problem. Second, we analyze the general complexities that can appear for each solution notion separately and relate these classes of multivalued functions to other known classes, extending Figure 3.1 from page 43.

We show that there are natural problems where the complexity of the various solution notions can differ drastically, unless P = NP (cf. Table 5.1). Furthermore, the diversity in the complexities of the solution notions for a general single problem can be vast: We show that for all sets $A, B, C \in \mathrm{NP}$ such that $A \leq_{\mathrm{T}}^{\mathrm{p}} B \leq_{\mathrm{T}}^{\mathrm{p}} C$ there is a multiobjective problem $\mathcal{O}$ such that $A \equiv_{\mathrm{T}}^{\mathrm{p}}$ A-$\mathcal{O}$, $B \equiv_{\mathrm{T}}^{\mathrm{p}}$ L-$\mathcal{O}$ and $C \equiv_{\mathrm{T}}^{\mathrm{p}}$ D-$\mathcal{O}$. Furthermore, the notions W-$\mathcal{O}$ and D-$\mathcal{O}$ always have to satisfy a certain restriction that gets stricter the easier D-$\mathcal{O}$ gets.

For linear problems, this is not possible, though: We show that the notions W-$\mathcal{O}$, L-$\mathcal{O}$ and A-$\mathcal{O}$ are all equivalent and furthermore, they are equivalent to the single-objective variant of the problem.

Interestingly, it turns out that for general problems the solution notions A-$\mathcal{O}$ and Val(A-$\mathcal{O}$), that are the simplest notions for a fixed problem, cover the largest range of complexities. With the exception of the arbitrary optimum notions, all search notions are equivalent to some set in NP and vice-versa and all value notions are equivalent to some function in PMV and vice-versa. In contrast, every function in NPMV is equivalent to Val(A-$\mathcal{O}$) for a suitable $\mathcal{O}$ and the class of A-$\mathcal{O}$-functions lies between PMV and $\mathrm{coNPMV_t}$ with respect to the $\subseteq_{\equiv_{\mathrm{T}}^{\mathrm{p}}}$-relation.

The results obtained in this chapter are summarized in Figure 5.1 on page 69 and Figure 5.3 on page 80.

5.1 Reducibilities and Evidence Against Them

For every fixed multiobjective problem $\mathcal{O}$, we investigate the reducibility among search and value notions for $\mathcal{O}$. More specifically, for every possible combination we either show that the reducibility holds for all multiobjective problems (Theorem 5.1) or we give evidence for the existence of a counter example (Theorem 5.10, Corollary 5.12). We end

up with a taxonomy (Figure 5.1 on page 69), which is complete in the sense that all possible combinations are covered.

Theorem 5.1. Let $\mathcal{O} = (S, f, \geq)$ be some k-objective problem.

1. $\mathrm{Val}(\mathcal{X}\text{-}\mathcal{O}) \leq_{\mathrm{T}}^{\mathrm{p}} \mathcal{X}\text{-}\mathcal{O}$ for $\mathcal{X} \in \{\mathrm{A}, \mathrm{L}, \mathrm{S}, \mathrm{D}, \mathrm{C}_1, \mathrm{C}_2, \ldots, \mathrm{C}_k, \mathrm{W}\}$
2. $\mathrm{A}\text{-}\mathcal{O} \leq_{\mathrm{T}}^{\mathrm{p}} \mathrm{L}\text{-}\mathcal{O} \leq_{\mathrm{T}}^{\mathrm{p}} \mathrm{S}\text{-}\mathcal{O}$ and $\mathrm{Val}(\mathrm{A}\text{-}\mathcal{O}) \leq_{\mathrm{T}}^{\mathrm{p}} \mathrm{Val}(\mathrm{L}\text{-}\mathcal{O}) \leq_{\mathrm{T}}^{\mathrm{p}} \mathrm{Val}(\mathrm{S}\text{-}\mathcal{O})$
3. $\mathrm{S}\text{-}\mathcal{O} \equiv_{\mathrm{T}}^{\mathrm{p}} \mathrm{D}\text{-}\mathcal{O} \equiv_{\mathrm{T}}^{\mathrm{p}} \mathrm{C_i}\text{-}\mathcal{O}$ and $\mathrm{Val}(\mathrm{S}\text{-}\mathcal{O}) \equiv_{\mathrm{T}}^{\mathrm{p}} \mathrm{Val}(\mathrm{D}\text{-}\mathcal{O}) \equiv_{\mathrm{T}}^{\mathrm{p}} \mathrm{Val}(\mathrm{C_i}\text{-}\mathcal{O})$ for all $i \in \{1, \ldots, k\}$
4. $\mathrm{L}\text{-}\mathcal{O} \leq_{\mathrm{T}}^{\mathrm{p}} \mathrm{W}\text{-}\mathcal{O}$ and $\mathrm{Val}(\mathrm{L}\text{-}\mathcal{O}) \leq_{\mathrm{T}}^{\mathrm{p}} \mathrm{Val}(\mathrm{W}\text{-}\mathcal{O})$
5. $\mathrm{D}\text{-}\mathcal{O}, \mathrm{W}\text{-}\mathcal{O} \leq_{\mathrm{T}}^{\mathrm{p}} \textsc{Sat}$

Proof.

1. Because of the definition of the value notions, it suffices to query $\mathcal{X}\text{-}\mathcal{O}$ on the input and return $f(s)$ if the answer is $s \neq \bot$. If the oracle answers $\bot$, return $\bot$.
2. Any solution of $\mathrm{L}\text{-}\mathcal{O}$ is an optimal solution and thus solves $\mathrm{A}\text{-}\mathcal{O}$. To solve $\mathrm{L}\text{-}\mathcal{O}$ we use $\mathrm{S}\text{-}\mathcal{O}$ to perform a binary search that respects the priority of objectives given in $\mathrm{L}\text{-}\mathcal{O}$ (i. e., we first optimize the objectives with the higher priority and proceed by optimizing the lower prioritized objectives, while forcing the optimal values for objectives with higher priority).

   Since this proof did not exploit any properties of the solutions but only of their values, it is also valid for the respective value notions.
3. First observe that $\mathrm{D}\text{-}\mathcal{O} \equiv_{\mathrm{T}}^{\mathrm{p}} \mathrm{S}\text{-}\mathcal{O}$, since a solution to $\mathrm{S}\text{-}\mathcal{O}$ is also a solution to $\mathrm{D}\text{-}\mathcal{O}$, whereas a binary search on $\mathrm{D}\text{-}\mathcal{O}$ also solves $\mathrm{S}\text{-}\mathcal{O}$. Now, suppose we want to solve $\mathrm{C_i}\text{-}\mathcal{O}$. A binary search over objective i that keeps the other objectives fixed to their constrained values shows $\mathrm{C_i}\text{-}\mathcal{O} \leq_{\mathrm{T}}^{\mathrm{p}} \mathrm{D}\text{-}\mathcal{O}$. On the other hand, optimizing objective i with constraints $b_j = c_j$ for all $j \neq i$ where $c = (c_1, \ldots, c_k)$ is the input cost vector for $\mathrm{D}\text{-}\mathcal{O}$ shows $\mathrm{D}\text{-}\mathcal{O} \leq_{\mathrm{T}}^{\mathrm{p}} \mathrm{C_i}\text{-}\mathcal{O}$.

   Similar to the previous part, this again shows the assertions for the value notions.
4. Since f is polynomial-time computable, there is some polynomial p such that $f_i^x(s) < 2^{p(|x|)}$ for all x, i and $s \in S^x$. Given some instance x and a fixed order of objectives we use the weight $(2^{p(|x|)})^{k-1}$ for the objective with the highest priority, $(2^{p(|x|)})^{k-2}$ for the objective with the second highest priority and so on. This shows $\mathrm{L}\text{-}\mathcal{O} \leq_{\mathrm{T}}^{\mathrm{p}} \mathrm{W}\text{-}\mathcal{O}$ and also $\mathrm{Val}(\mathrm{L}\text{-}\mathcal{O}) \leq_{\mathrm{T}}^{\mathrm{p}} \mathrm{Val}(\mathrm{W}\text{-}\mathcal{O})$.
5. Observe that by Proposition 4.7, $\mathrm{D}\text{-}\mathcal{O} \in \mathrm{PMV}$ and f-$\textsc{Sat}$ and thus $\textsc{Sat}$ is $\leq_{\mathrm{T}}^{\mathrm{p}}$-hard for NPMV by Proposition 3.8. For $\mathrm{W}\text{-}\mathcal{O}$, define $g(\langle x, \langle \omega \rangle, a \rangle) = \{y \in S^x \mid \omega^{\mathrm{T}} f^x(y) \geq a\}$ and observe that $g \in \mathrm{NPMV}$. Furthermore, we have $\mathrm{W}\text{-}\mathcal{O} \leq_{\mathrm{T}}^{\mathrm{p}} g$ by binary search and thus $\mathrm{W}\text{-}\mathcal{O} \leq_{\mathrm{T}}^{\mathrm{p}} \textsc{Sat}$ by the same argumentation as before. □

We will see that Theorem 5.1 is complete in the sense that no other reductions between solution notions hold assuming reasonable complexity-theoretic assumptions.

Furthermore, it is not uncommon that for a fixed problem, the hardness of the search notions differ considerably. Below we give several examples of natural problems that

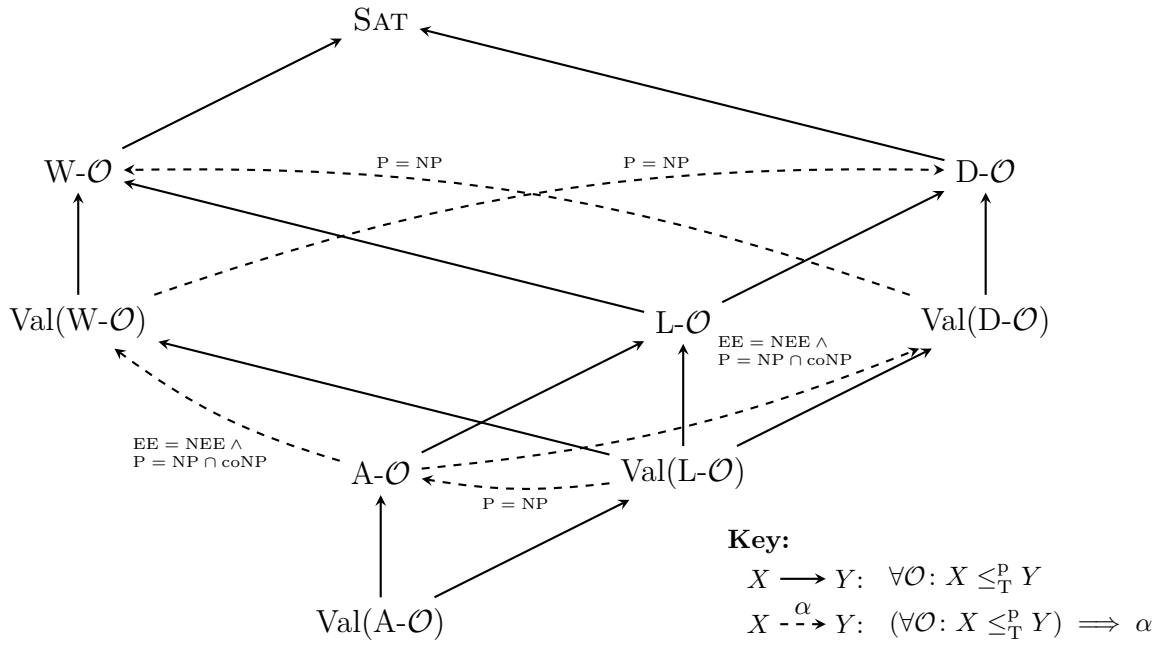


Figure 5.1: Complete taxonomy of reductions between search and value notions. Non-dashed arrows indicate reducibility for all problems $\mathcal{O}$, whereas dashed arrows provide evidence against such a general reducibility. Observe that such evidence propagates along non-dashed arrows (arrow heads backwards and arrow tails forwards) and we hence have evidence against all remaining possible reductions. Further note that D-$\mathcal{O}$ $\equiv^{\mathrm{p}}_{\mathrm{T}}$ S-$\mathcal{O}$ $\equiv^{\mathrm{p}}_{\mathrm{T}}$ $\mathrm{C_i}$-$\mathcal{O}$ and Val(D-$\mathcal{O}$) $\equiv^{\mathrm{p}}_{\mathrm{T}}$ Val(S-$\mathcal{O}$) $\equiv^{\mathrm{p}}_{\mathrm{T}}$ Val($\mathrm{C_i}$-$\mathcal{O}$) for $i \in \{1, \ldots, k\}$.

are NP-hard with respect to one notion and that are polynomial-time solvable with respect to another one. The results are summarized in Table 5.1 on page 72. This shows the importance of an exact specification of the NP-hardness notion that is used when discussing the complexity of multiobjective problems.

For so-called linear multiobjective problems $\mathcal{O}$, it is often the case that D-$\mathcal{O}$ is NP-hard because of a reduction from Knapsack and all other solution notions are solvable in polynomial time. Below we show that for such problems, all search notions apart from D-$\mathcal{O}$ must be equivalent and all value notions apart from Val(D-$\mathcal{O}$) must be equivalent. Papadimitriou and Yannakakis [PY00] gave a detailed analysis of linear multiobjective problems, where they mostly studied their approximability.

Definition 5.2. A k-objective problem $\mathcal{O} = (S, f, \leftarrow)$ is called *linear* if

- its instances are $\langle x', m, \langle A \rangle \rangle$ for $x', m \in \mathbb{N}$ and $A \in \mathbb{N}^{k \times m}$,
- the solutions are vectors $S^{\langle x', m, \langle A \rangle \rangle} \subseteq \mathbb{N}^m$ where $S^{\langle x', m, \langle A \rangle \rangle}$ does not depend on A, i. e. $S^{\langle x', m, \langle A \rangle \rangle} = S^{\langle x', m, \langle B \rangle \rangle}$ for all $A, B \in \mathbb{N}^{k \times m}$ and
- $f^{\langle x', m, \langle A \rangle \rangle}(s) = A \cdot s$.

So for a linear problem, it is important that any linear function of the solution vector is a valid objective function for some instance of the problem, i. e. an instance can be equipped with any possible matrix $A \in \mathbb{N}^{k \times m}$.

Proposition 5.3. If $\mathcal{O} = (S, f, \leftarrow)$ for $\leftarrow \in \{\leq, \geq\}$ is a linear multiobjective problem, then

$$\text{W-}\mathcal{O} \equiv^{\text{p}}_{\text{T}} \text{L-}\mathcal{O} \equiv^{\text{p}}_{\text{T}} \text{A-}\mathcal{O} \equiv^{\text{p}}_{\text{T}} \text{A-}(S, f_1, \leftarrow_1)$$

and

$$\text{Val}(\text{W-}\mathcal{O}) \equiv^{\text{p}}_{\text{T}} \text{Val}(\text{L-}\mathcal{O}) \equiv^{\text{p}}_{\text{T}} \text{Val}(\text{A-}\mathcal{O}) \equiv^{\text{p}}_{\text{T}} \text{Val}(\text{A-}(S, f_1, \leftarrow_1)).$$

Proof. Let $\mathcal{O} = (S, f, \leftarrow)$ be a linear k-objective problem and let $\mathcal{O}_1 = (S, f_1, \leftarrow_1)$ be its single-objective variant. For the search notions, it suffices to show W-$\mathcal{O} \leq^{\text{p}}_{\text{T}}$ A-$\mathcal{O}_1 \leq^{\text{p}}_{\text{T}}$ A-$\mathcal{O}$ by Theorem 5.1. The proof is the same for the value notions.

For W-$\mathcal{O}$, on input of $\langle x, \langle \omega \rangle \rangle$, $\omega \in \mathbb{N}^k$, we have to optimize the function $s \mapsto \omega^{\mathrm{T}} A s$, where $x = \langle x', m, \langle A \rangle \rangle$. A reduction to A-$\mathcal{O}_1$ can be accomplished by a query to $\langle x', m, \langle \omega^{\mathrm{T}} A \rangle \rangle$. For the second reduction, on input $\langle x', m, \langle A \rangle \rangle$, we query $\langle x', m, A' \rangle$, where $A' = (1, 0, \ldots, 0)^{\mathrm{T}} A$. □

Proposition 5.4. For any $k \geq 1$, the problems k-MM, k-MST and k-SP are linear k-objective problems (under a suitable instance and solution encoding). Furthermore, for any $\mathcal{O} \in \{k\text{-MM}, k\text{-SP}, k\text{-MST}\}$,

1. W-$\mathcal{O}$ is polynomial-time solvable.
2. D-$\mathcal{O}$ is polynomial-time solvable if $k = 1$ and Val(D-$\mathcal{O}$) (and thus D-$\mathcal{O}$) is NP-hard if $k \geq 2$.

Proof. The first statement is an immediate consequence of Proposition 5.3, because the single-objective variants of all three problems are solvable in polynomial time.

For $k = 1$, we have D-$\mathcal{O} \equiv^{\text{p}}_{\text{T}}$ W-$\mathcal{O}$ by Proposition 4.4 and thus D-$\mathcal{O}$ is polynomial-time solvable by the first statement. A straightforward reduction from Knapsack shows the NP-hardness of Val(D-$\mathcal{O}$) for $k \geq 2$ for all three problems $\mathcal{O} =$ 2-MST [AAN82], 2-SP [HZ80], 2-MM. □

Proposition 5.5. For any $k \geq 1$, $\mathcal{O} = k$-TSP is a linear k-objective problem (under a suitable instance and solution encoding) and Val(A-$\mathcal{O}$) is NP-hard (for any $k \geq 1$).

Proof. The NP-hardness of the usual single-objective traveling salesperson problem carries over to multiple objectives by Proposition 5.3. □

Proposition 5.6. Let $\mathcal{O} =$ 2-QDE.

1. D-$\mathcal{O}$ is polynomial-time solvable.
2. Val(W-$\mathcal{O}$) is NP-hard.

Proof.

1. C_1-$\mathcal{O}$ is the single-objective problem that on input of $a, b, c, b_2 \in \mathbb{N}$ searches for the smallest $x \in \mathbb{N}$ such that $\exists y \in \mathbb{N}[y^2 \leq b_2 \wedge ax^2 + by^2 - c \geq 0]$. Note that in the case of b_2 not being a square, replacing it by the greatest square smaller than b_2 does not change the problem. So we can assume b_2 to be a square. If $c \leq bb_2$, we obviously have $x = 0$, and if $c > bb_2$ and $a = 0$, there is no solution. Otherwise, we have $c > bb_2$ and $a > 0$, which directly implies $x = \left\lceil \sqrt{\frac{c - bb_2}{a}} \right\rceil$. Since all computations can be carried out and all cases can be distinguished efficiently, C_1-$\mathcal{O}$ is polynomial-time solvable. The assertion follows by Theorem 5.1.

2. The set QDE $= \{(a,b,c) \in \mathbb{N} \mid \exists x,y \in \mathbb{N}(ax^2+by^2-c=0)\}$ is NP-complete [MA78]. We reduce QDE to Val(W-$\mathcal{O}$). For given (a,b,c) we solve Val(W-$\mathcal{O}$) for the weight vector $w=(a,b)$. If no solution is found, then $S^{\langle a,b,c\rangle}=\emptyset$ and hence $(a,b,c) \notin$ QDE. Otherwise, let (x^2,y^2) be the solution of Val(W-$\mathcal{O}$), i. e., $x,y \in \mathbb{N}$ such that $ax^2+by^2-c \geq 0$ and ax^2+by^2 is minimal. It follows that $(a,b,c) \in$ QDE if and only if $ax^2+by^2-c=0$. This shows the NP-hardness of Val(W-$\mathcal{O}$). □

Proposition 5.7 ([HvdV90])**.** For $\mathcal{O}$ = 2-LF, W-$\mathcal{O}$ and D-$\mathcal{O}$ are polynomial-time solvable.

Proof. Hoogeveen and van de Velde [HvdV90] show that for $(S, f, \leftarrow)$ = 2-LF the number of Pareto-optimal points is polynomially bounded and that there exists a polynomial-time algorithm that on input x computes some $S \subseteq S^x_{\text{opt}}$ such that $f^x(S) = f^x(S^x_{\text{opt}})$ [HvdV90, Theorem 8], [Hoo92, page 15]. □

Proposition 5.8 ([Bak74, Hoo92])**.** Let $\mathcal{O}$ = 2-LWF, let L_1-$\mathcal{O}$ be the solution notion that first minimizes the maximum lateness, and L_2-$\mathcal{O}$ the notion that first minimizes the weighted flowtime.

1. Val(L_1-$\mathcal{O}$) is NP-hard.

2. L_2-$\mathcal{O}$ is polynomial-time solvable.

Proof.

1. Scheduling problems are often denoted by the three-field notation scheme $\alpha|\beta|\gamma$ introduced by Graham et al. [GLLK79], where α describes the machine environment, β the job constraints, and γ the objective function. In this notation, the problem L_1-$\mathcal{O}$ is written as $1||F_h(L_{\max}, \sum w_jC_j)$, where the F_h indicates that the optimization is hierarchical such that $L_{\max}$ is the primary and $\sum w_jC_j$ the secondary objective. Hoogeveen [Hoo92, Theorem 11] shows the NP-hardness of the decision variant of $1||F_h(L_{\max}, \sum w_jC_j)$.

2. The problem $1||F_h(\sum w_jC_j, L_{\max})$ is solved in time $O(n \log n)$ by sequencing the jobs in non-decreasing order of the ratios p_i/w_i (which minimizes the weighted flowtime) such that ties are broken by sequencing the jobs in non-decreasing order of their due dates (which minimizes the maximum lateness) [Bak74], [Hoo92, page 17]. Hence L_2-$\mathcal{O}$ is polynomial-time solvable. □

As a consequence of the analysis of these problems, under the assumption P ≠ NP we can prove the strictness of all Turing-reducibilities between search notions shown in Theorem 5.1 and depicted in Figure 5.1 on page 69 and some reducibilities between value and search notions.

Corollary 5.9. If P ≠ NP then there exist 2-objective problems $\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3, \mathcal{O}_4, \mathcal{O}_5$ where all objectives have to be minimized such that the following holds.

1. L-$\mathcal{O}_1 \not\leq^{\text{p}}_{\text{T}}$ A-$\mathcal{O}_1$

2. W-$\mathcal{O}_2 \not\leq^{\text{p}}_{\text{T}}$ L-$\mathcal{O}_2$

Problem $\mathcal{O}$	A-$\mathcal{O}$	L_1-$\mathcal{O}$	L_2-$\mathcal{O}$	W-$\mathcal{O}$	S-$\mathcal{O}$,D-$\mathcal{O}$,C_i-$\mathcal{O}$	Ref.
2-LF	P	P	P	P	P	Prop. 5.7
2-MM	P	P	P	P	NP-hard	Prop. 5.4
2-SP	P	P	P	P	NP-hard	Prop. 5.4
2-MST	P	P	P	P	NP-hard	Prop. 5.4
2-QDE	P	P	P	NP-hard	P	Prop. 5.6
2-LWF	P	NP-hard	P	NP-hard	NP-hard	Prop. 5.8
2-TSP	NP-hard	NP-hard	NP-hard	NP-hard	NP-hard	Prop. 5.5

Table 5.1: Separation of NP-hardness notions for multiobjective problems. For these examples, the value notions are always equivalent to their search counterparts. "P" indicates that this solution notion for the problem is polynomial-time solvable. L_i-$\mathcal{O}$ denotes the lexicographical problem where the i-th objective is the primary one.

3. D-$\mathcal{O}_3 \not\leq^{\mathrm{p}}_{\mathrm{T}}$ L-$\mathcal{O}_3$
4. D-$\mathcal{O}_4 \not\leq^{\mathrm{p}}_{\mathrm{T}}$ W-$\mathcal{O}_4$
5. W-$\mathcal{O}_5 \not\leq^{\mathrm{p}}_{\mathrm{T}}$ D-$\mathcal{O}_5$

Theorem 5.10. If P $\neq$ NP, then there exist two-objective NP optimization problems $\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3$ such that:

1. Val(L-$\mathcal{O}_1$) $\not\leq^{\mathrm{p}}_{\mathrm{T}}$ A-$\mathcal{O}_1$
2. Val(W-$\mathcal{O}_2$) $\not\leq^{\mathrm{p}}_{\mathrm{T}}$ D-$\mathcal{O}_2$
3. Val(D-$\mathcal{O}_3$) $\not\leq^{\mathrm{p}}_{\mathrm{T}}$ W-$\mathcal{O}_3$

What remains is to find evidence against a possible reduction from a search notion to a value notion, for instance from A-$\mathcal{O}$ to Val(W-$\mathcal{O}$). This question is related to the study of search versus decision [BD76, Bal89, BBFG91], more precisely to the notion of functional self-reducibility, which was introduced by Borodin and Demers [BD76]. A problem is *functionally self-reducible* if it belongs to the following set.

$$\mathrm{SRD}_\forall = \{A \in \mathrm{NP} \mid \text{ for all } g \in \mathrm{PMV} \text{ such that } A = \mathrm{dom}(g) \text{ it holds that } g \leq^{\mathrm{p}}_{\mathrm{T}} A\}$$

The name of this class indicates that functional self-reducibility is a universal variant of the notion of search reduces to decision: We say that *search reduces to decision* for a problem $A \in \mathrm{NP}$ if there is some $g \in \mathrm{PMV}$ such that $A = \mathrm{dom}(g)$ and $g \leq^{\mathrm{p}}_{\mathrm{T}} A$. Statement 1 in the following theorem is equivalent to the statement NP $\neq \mathrm{SRD}_\forall$. Moreover, if there exists an $L \in \mathrm{NP}$ for which search does not reduce to decision (as shown by Beigel et al. [BBFG91] under the assumption EE $\neq$ NEE), then statement 1 holds.

Theorem 5.11. The following statements are equivalent:

1. There is some $g \in \mathrm{PMV}$ such that $g \not\leq^{\mathrm{p}}_{\mathrm{T}} \mathrm{dom}(g)$.
2. There is a multiobjective problem $\mathcal{O} = (S, f, \geq)$ such that A-$\mathcal{O} \not\leq^{\mathrm{p}}_{\mathrm{T}}$ Val(W-$\mathcal{O}$) $\equiv^{\mathrm{p}}_{\mathrm{T}}$ Val(D-$\mathcal{O}$) and f is constant.

Proof. "1 $\Rightarrow$ 2": Define $\mathcal{O} = (S, f, \geq)$ by $S^x = g(x)$ and $f(\langle x, y\rangle) = 1$ for $y \in S^x$. So ($x \in \mathrm{dom}(g) \iff S^x \neq \emptyset$) and hence $\mathrm{dom}(g) \equiv^{\mathrm{p}}_{\mathrm{T}} \mathrm{Val}(\text{W-}\mathcal{O}) \equiv^{\mathrm{p}}_{\mathrm{T}} \mathrm{Val}(\text{D-}\mathcal{O})$. The implication follows, since $\text{A-}\mathcal{O} = g \not\leq^{\mathrm{p}}_{\mathrm{T}} \mathrm{dom}(g)$.

"2 $\Rightarrow$ 1": From f being constant it follows that each $y \in S^x$ is optimal and $\text{A-}\mathcal{O}(x) = S^x$. Hence, for $g := \text{A-}\mathcal{O}$ it holds that $g \in \mathrm{PMV}$ (by the definition of multiobjective problems). Moreover, $x \in \mathrm{dom}(g) \iff \mathrm{Val}(\text{W-}\mathcal{O})(\langle x, \langle 1, \ldots, 1\rangle\rangle) \neq \emptyset$ and hence $\mathrm{dom}(g) \leq^{\mathrm{p}}_{\mathrm{T}} \mathrm{Val}(\text{W-}\mathcal{O})$. Therefore, $g \not\leq^{\mathrm{p}}_{\mathrm{T}} \mathrm{dom}(g)$, since otherwise $\text{A-}\mathcal{O} \leq^{\mathrm{p}}_{\mathrm{T}} \mathrm{dom}(g) \leq^{\mathrm{p}}_{\mathrm{T}} \mathrm{Val}(\text{W-}\mathcal{O})$. □

Corollary 5.12. If $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$ or $\mathrm{EE} \neq \mathrm{NEE}$, then there exists a multiobjective problem $\mathcal{O} = (S, f, \geq)$ such that $\text{A-}\mathcal{O} \not\leq^{\mathrm{p}}_{\mathrm{T}} \mathrm{Val}(\text{W-}\mathcal{O}) \equiv^{\mathrm{p}}_{\mathrm{T}} \mathrm{Val}(\text{D-}\mathcal{O})$.

Proof. Valiant [Val76, Proposition 5] shows that $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$ implies statement 1 in Theorem 5.11. Beigel et al. [BBFG91] show that $\mathrm{EE} \neq \mathrm{NEE}$ implies the same statement (cf. Theorem 3.25). □

5.2 Complexities of Value Notions of a Single Problem

In the previous section we have seen that for any two search notions X-$\mathcal{O}$, Y-$\mathcal{O}$ such that $\text{X-}\mathcal{O} \leq^{\mathrm{p}}_{\mathrm{T}} \text{Y-}\mathcal{O}$, there indeed is a problem $\mathcal{O}$ where Y-$\mathcal{O}$ is NP-hard and X-$\mathcal{O}$ is polynomial time solvable. For the value notions, we show an even stronger result. We will see that almost any combination of complexities can occur:

For all sets $A, L, D, W \in \mathrm{NP}$ that satisfy the following moderate requirements there exist a multiobjective problem $\mathcal{O}$ whose value notions $\mathrm{Val}(\text{A-}\mathcal{O})$, $\mathrm{Val}(\text{L-}\mathcal{O})$, $\mathrm{Val}(\text{D-}\mathcal{O})$, $\mathrm{Val}(\text{W-}\mathcal{O})$ are equivalent to A, L, D, W, respectively.

- Requirement 1: $A \leq^{\mathrm{p}}_{\mathrm{T}} L \leq^{\mathrm{p}}_{\mathrm{T}} D$ and $L \leq^{\mathrm{p}}_{\mathrm{T}} W$
- Requirement 2: $W \equiv^{\mathrm{p}}_{\mathrm{T}} g$ for some $g \in \max{\cdot}\, D$

The first requirement is necessary, since by Theorem 5.1 these reducibilities hold for all multiobjective problems. The necessity of the second requirement is shown by Proposition 5.15 and discussed at the end of this section. Interestingly, Requirement 2 does not impose an upper bound on the complexity of W but rather a restriction on the possible Turing-degrees it can lie in.

Note that here, we are only concerned with complexities that can be expressed in terms of sets. The complexities that can occur in terms of multivalued functions will be addressed in the next section for each notion individually.

Theorem 5.13. Let A, L, D, $W \in \mathrm{NP}$ such that $A \leq^{\mathrm{p}}_{\mathrm{T}} L \leq^{\mathrm{p}}_{\mathrm{T}} D$ and $L \leq^{\mathrm{p}}_{\mathrm{T}} W \equiv^{\mathrm{p}}_{\mathrm{T}} g$ for some $g \in \max{\cdot}\, D$. Then there exists a two-objective problem $\mathcal{O} = (S, f, \geq)$ such that

1. $\mathrm{Val}(\text{A-}\mathcal{O}) \equiv^{\mathrm{p}}_{\mathrm{T}} A$
2. $\mathrm{Val}(\text{L-}\mathcal{O}) \equiv^{\mathrm{p}}_{\mathrm{T}} L$
3. $\mathrm{Val}(\text{D-}\mathcal{O}) \equiv^{\mathrm{p}}_{\mathrm{T}} D$
4. $\mathrm{Val}(\text{W-}\mathcal{O}) \equiv^{\mathrm{p}}_{\mathrm{T}} W$

Proof. Let A, L, D, $W \in \mathrm{NP}$, $g \in \max\cdot D$ with reduction relations as required in the statement of the theorem and let A_w, L_w, $D_w \in \mathrm{P}$ be corresponding witness sets. For the order of objectives with regard to Val(L-$\mathcal{O}$) we choose to give priority to the first objective.

The idea of the proof is to construct a two-objective problem where the solutions are arranged in three stages: In the stage for Val(A-$\mathcal{O}$) and A the witnesses for $x \in A$ are encoded as solutions with a constant value. The second stage handles Val(L-$\mathcal{O}$) and L: Here, we also encode the witnesses into solutions but ignore them when computing the value. We also add a trivial optimal solution (though not lexicographically optimal) such that Val(A-$\mathcal{O}$) can solve this stage. The third stage is the most complicated stage: We again add trivial solutions for Val(A-$\mathcal{O}$) and Val(L-$\mathcal{O}$) and encode the witnesses for D into the solutions. Furthermore, the value of all solutions corresponding to $\langle x, i\rangle \in D$ lies on a diagonal at position i. This way we achieve that Val(W-$\mathcal{O}$) can compute the maximal i such that $\langle x, i\rangle \in D$ and nothing more and we still have Val(D-$\mathcal{O}$) $\equiv^{\mathrm{p}}_{\mathrm{T}} D$.

We first show that we can demand the following without loss of generality:

Claim 5.14. By replacing all sets and functions in the theorem with equivalent sets and functions, it can be assumed that p is a polynomial such that for any x the following holds:

1. For any $X_w \in \{A_w, L_w, D_w\}$ and any y such that $\langle x, y\rangle \in X_w$ it holds that $y < 2^{p(|x|)}$.
2. For all y where $\langle x, y\rangle \in D$ it holds that $0 < y < 2^{p(|x|)} - 1$ and $g = \max_p\cdot D$.
3. There is at least one y such that $\langle x, y\rangle \in D$.
4. For all y it holds that $\langle x, y\rangle \in D \iff \langle x, 2^{p(|x|)} - 1 - y\rangle \in D$.

Proof of the claim. Statement 1 can be fulfilled by using a large enough polynomial and removing witnesses from the witness set that are too large. Note that 1 remains fulfilled for larger polynomials.

For an arbitrary $D_0 \in \mathrm{NP}$ and $g_0 = \max_{p_0}\cdot D_0$ (with $p_0 > 0$), we now construct $D \equiv^{\mathrm{p}}_{\mathrm{T}} D_0$ and $g = \max_p\cdot D \equiv^{\mathrm{p}}_{\mathrm{T}} g_0$ for some polynomial p that fulfill the assertions. Consider the set

$$\begin{aligned} D' :=& \{\langle\langle x, 0\rangle, y\rangle \mid \langle x, y\rangle \in D_0 \text{ and } y < 2^{p_0(|x|)}\} \cup \\ & \{\langle\langle x, 1 + y\rangle, a\rangle \mid a = 1 \vee (a = 0 \wedge \langle x, y\rangle \in D_0)\}. \end{aligned}$$

Observe that $D_0 \equiv^{\mathrm{p}}_{\mathrm{T}} D'$, $g_0 \equiv^{\mathrm{p}}_{\mathrm{T}} g' := \max_{p_0}\cdot D'$ and for all $\langle x, y\rangle \in D'$ it holds that $y < 2^{p_0(|x|)}$. Choose some polynomial p such that $p > p_0 + 3$ and p is large enough for assertion 1. Observe that for

$$\begin{aligned} D := & \{\langle x, 2^{p(|x|)-1} + y\rangle \mid \langle x, y\rangle \in D'\} \cup \\ & \{\langle x, 2^{p(|x|)-1} - 1\rangle \mid x \in \mathbb{N}\} \cup \\ & \{\langle x, 2^{p(|x|)-1}\rangle \mid x \in \mathbb{N}\} \cup \\ & \{\langle x, 2^{p(|x|)-1} - 1 - y\rangle \mid \langle x, y\rangle \in D'\} \end{aligned}$$

it holds that $D \equiv^{\mathrm{p}}_{\mathrm{T}} D'$ and $g := \max_p\cdot D \equiv^{\mathrm{p}}_{\mathrm{T}} g_0$. Moreover, D and g fulfill the remaining assertions. ◇

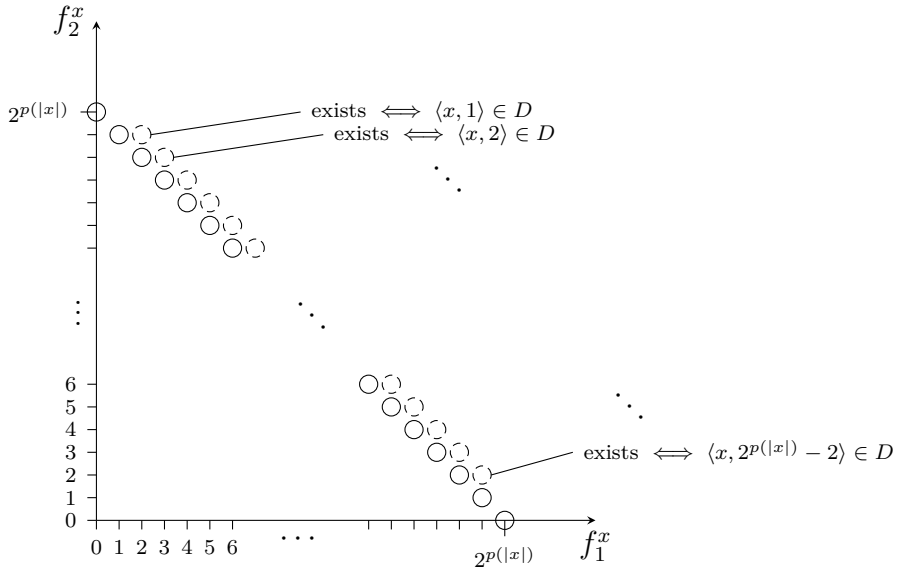


Figure 5.2: Illustration of $f(S^{3x+2})$ (Proof of Theorem 5.13).

We define the 2-objective maximization problem $\mathcal{O} = (S, f, \geq)$ by (cf. Figure 5.2)

$$\begin{aligned} S^{3x} &= \{\langle 0, 0, y\rangle \mid \langle x, y\rangle \in A_w\} && \text{(stage for } A\text{)} \\ S^{3x+1} &= \{\langle 0, 0, 0\rangle\} \cup \{\langle 0, 1, y\rangle \mid \langle x, y\rangle \in L_w\} && \text{(stage for } L\text{)} \\ S^{3x+2} &= \{\langle 0, i, 0\rangle \mid i \leq 2^{p(|x|)}\} \cup && \text{(stage for } W \text{ and } D\text{)} \\ &\quad \{\langle 1, y, z\rangle \mid y < 2^{p(|x|)} \text{ and } \langle\langle x, y\rangle, z\rangle \in D_w\} \\ f^{3x+r}(\langle a, i, z\rangle) &= (i + a, j) \text{ such that } i + j = 2^{p(|x|)} \end{aligned}$$

The lengths of valid solutions are obviously polynomially bounded and $S \in \mathrm{P}$, because $\langle a, i, z\rangle \in S^{3x+r}$ can always be checked by simple arithmetic and optionally some query to a witness set in P. The objective function f is computable in polynomial time. We now verify the stated equivalences.

1. Val(A-$\mathcal{O}$) $\leq_{\mathrm{T}}^{\mathrm{p}}$ A: Note that the value $(0, 2^{p(|x|)})$ is always optimal for instances of the form $3x + 1$ or $3x + 2$, so the reduction algorithm can output it without querying A. For instances of the form $3x$ it queries A for x and outputs $(0, 2^{p(|x|)})$ if the answer is yes and $\perp$ otherwise.

   $A \leq_{\mathrm{T}}^{\mathrm{p}}$ Val(A-$\mathcal{O}$): Here, on input x the reduction is done by a query for Val(A-$\mathcal{O}$)$(3x)$ with output "no" if and only if the answer is $\perp$.

2. Val(L-$\mathcal{O}$) $\leq_{\mathrm{T}}^{\mathrm{p}}$ L: Note that for instances of the form $3x + 2$, the values $(0, 2^{p(|x|)})$ and $(2^{p(|x|)}, 0)$ are always optimal, so the reduction algorithm can output a lexicographically optimal solution without querying L. Instances of the form $3x$ can be solved by a query to Val(A-$\mathcal{O}$) $\leq_{\mathrm{T}}^{\mathrm{p}}$ $A \leq_{\mathrm{T}}^{\mathrm{p}}$ L. Let now the instance be $3x + 1$. Note that Val(L-$\mathcal{O}$) has to output $(0, 2^{p(|x|)})$ if $x \notin L$ and $(1, 2^{p(|x|)} - 1)$ otherwise, which can be checked by a simple query to L.

   $L \leq_{\mathrm{T}}^{\mathrm{p}}$ Val(L-$\mathcal{O}$): Similar to the case for A, the reduction is a simple query to Val(L-$\mathcal{O}$)$(3x + 1)$.

3. $\mathrm{Val}(\text{D-}\mathcal{O}) \leq_{\mathrm{T}}^{\mathrm{p}} D$: Instances not of the form $3x+2$ can be handled by queries to $\mathrm{Val}(\text{A-}\mathcal{O})$ or $\mathrm{Val}(\text{L-}\mathcal{O})$ since $\mathrm{Val}(\text{A-}\mathcal{O}) \leq_{\mathrm{T}}^{\mathrm{p}} A \leq_{\mathrm{T}}^{\mathrm{p}} D$ and $\mathrm{Val}(\text{L-}\mathcal{O}) \leq_{\mathrm{T}}^{\mathrm{p}} L \leq_{\mathrm{T}}^{\mathrm{p}} D$. Let now $\langle 3x+2, \langle i,j\rangle\rangle$ be the input.

   If $i+j \leq 2^{p(|x|)}$, output $f^{3x+2}(\langle 0,i,0\rangle) = (i, 2^{p(|x|)}-i)$, which is always the value of some solution. If $i+j > 2^{p(|x|)}+1$, there is no solution that (weakly) dominates this value, so output $\bot$. For the last case, $i+j = 2^{p(|x|)}+1$, note that the only solutions that can possibly (weakly) dominate the value (i,j) are those of type $\langle 1,y,z\rangle$ for $y < 2^{p(|x|)}$ and $\langle\langle x,y\rangle,z\rangle \in D_w$ which also have the value (i,j). This means that $y = i-1$, so we can return (i,j) if $\langle x, i-1\rangle \in D$ and $\bot$ otherwise.

   $D \leq_{\mathrm{T}}^{\mathrm{p}} \mathrm{Val}(\text{D-}\mathcal{O})$: On input $\langle x,y\rangle$, $\mathrm{Val}(\text{D-}\mathcal{O})(\langle 3x+2, \langle i,j\rangle\rangle)$ with $i = y+1$ and $j = 2^{p(|x|)} - y$ is queried. As shown in the previous paragraph, the result of this query tells whether or not $\langle x,y\rangle \in D$.

4. $\mathrm{Val}(\text{W-}\mathcal{O}) \leq_{\mathrm{T}}^{\mathrm{p}} W$: As in the case of $\mathrm{Val}(\text{D-}\mathcal{O})$, instances not of the form $3x+2$ can be handled by indirect reductions. For instances of the form $3x+2$ we show $\mathrm{Val}(\text{W-}\mathcal{O}) \leq_{\mathrm{T}}^{\mathrm{p}} g$:

   It obviously suffices to return values from the border of the convex hull of all solution values. It even suffices to consider only corner points of the convex hull. These corner points are $(0, 2^{p(|x|)})$, $(2^{p(|x|)}, 0)$, $(1+y_{\min}, 2^{p(|x|)} - y_{\min})$ and $(1+y_{\max}, 2^{p(|x|)} - y_{\max})$ where $y_{\min}$ and $y_{\max}$ are the minimal and maximal values for y such that $\langle x,y\rangle \in D$. Since we required that $\langle x,y\rangle \in D \iff \langle x, 2^{p(|x|)} - 1 - y\rangle \in D$, we only need to determine $y_{\max}$ and this can obviously be done by a query to $g(x)$ (note that we also required that there is at least one y such that $\langle x,y\rangle \in D$).

   $W \leq_{\mathrm{T}}^{\mathrm{p}} \mathrm{Val}(\text{W-}\mathcal{O})$: The reduction $g \leq_{\mathrm{T}}^{\mathrm{p}} \mathrm{Val}(\text{W-}\mathcal{O})$ holds as follows: On input x, $\mathrm{Val}(\text{W-}\mathcal{O})(\langle 3x+2, \langle w, w-1\rangle\rangle)$ for $w = 2^{p(|x|)}+1$ is queried. The weighted sum of the value of a solution $s = \langle a,i,z\rangle$ is

$$\begin{aligned} w f_1^{3x+2}(\langle a,i,z\rangle) + (w-1) f_2^{3x+2}(\langle a,i,z\rangle) &= w(i+a) + (w-1)(2^{p(|x|)} - i) \\ &= i + wa + (w-1)2^{p(|x|)}. \end{aligned}$$

   Since every possible value for i is at most $2^{p(|x|)} < w$ and we required that there is at least one y such that $\langle x,y\rangle \in D$, the function $\mathrm{Val}(\text{W-}\mathcal{O})$ returns the value of a solution of type $\langle 1,y,z\rangle$ with maximal y, which is exactly $g(x)$. □

We now show that in Theorem 5.13 it is necessary to restrict the relationship between D and W such that $W \equiv_{\mathrm{T}}^{\mathrm{p}} g$ for some $g \in \max{\cdot}\, D$. As a consequence, the complexities for $\mathrm{Val}(\text{A-}\mathcal{O})$, $\mathrm{Val}(\text{L-}\mathcal{O})$, $\mathrm{Val}(\text{D-}\mathcal{O})$, and $\mathrm{Val}(\text{W-}\mathcal{O})$ provided by Theorem 5.13 are indeed all possible complexities for the value notions that can be described in terms of sets (cf. Corollary 5.16).

Proposition 5.15. For every multiobjective problem $\mathcal{O} = (S, f, \geq)$ there is some $A \in \mathrm{NP}$ and $g \in \max{\cdot}\, A$ such that

$$\mathrm{Val}(\text{D-}\mathcal{O}) \equiv_{\mathrm{T}}^{\mathrm{p}} A \text{ and } \mathrm{Val}(\text{W-}\mathcal{O}) \equiv_{\mathrm{T}}^{\mathrm{p}} g.$$

Proof. For the k-objective problem $\mathcal{O} = (S, f, \geq)$ let

$$A := \{\langle\langle x, \langle w\rangle\rangle, \langle z, y_k, \ldots, y_1\rangle'\rangle \mid w \in \mathbb{N}^k, y_i < 2^{p(|x|)}, z = \sum_{i=1}^{k} w_i y_i, \text{ and there is some } s \in S^x \text{ such that } f^x(s) \geq (y_1, \ldots, y_k)\}$$

where p is a polynomial upper bound for all polynomials in the definition of $\mathcal{O}$ and $\langle z, y_k, \ldots, y_1\rangle' := 1 + z \cdot 2^{k \cdot p(|x|)} + \sum_{i=1}^{k} y_i \cdot 2^{(i-1)\cdot p(|x|)}$ for $z \in \mathbb{N}$ and $0 \leq y_i < 2^{p(|x|)}$. This means that $\langle\cdot\rangle'$ is a bijection between $\mathbb{N} \times \{0, \ldots, 2^{p(|x|)} - 1\}^k$ and $\mathbb{N}^+$ that transfers the lexicographical order on $\mathbb{N} \times \{0, \ldots, 2^{p(|x|)} - 1\}^k$ to the natural order on $\mathbb{N}^+$. Furthermore, for all $\langle\langle x, \langle w\rangle\rangle, \langle z, y_k, \ldots, y_1\rangle'\rangle \in A$ it holds that $\langle z, y_k, \ldots, y_1\rangle' < 2^{q(|\langle x, \langle w\rangle\rangle|)}$ for some polynomial q. Since $\{\langle x, s\rangle \mid x \in \mathbb{N}, s \in S^x\} \in \mathrm{P}$ and $f \in \mathrm{FP}$ we have $A \in \mathrm{NP}$. Let $g = \max_q \cdot A$. We will show $\mathrm{Val}(\text{D-}\mathcal{O}) \equiv_{\mathrm{T}}^{\mathrm{p}} A$ and $\mathrm{Val}(\text{W-}\mathcal{O}) \equiv_{\mathrm{T}}^{\mathrm{p}} g$.

1. $\mathrm{Val}(\text{D-}\mathcal{O}) \leq_{\mathrm{T}}^{\mathrm{p}} A$: On input $\langle x, \langle c\rangle\rangle$, we query $x' := \langle\langle x, \langle 0, \ldots, 0\rangle\rangle, \langle 0, c_k, \ldots, c_1\rangle'\rangle \in A$. If $x' \notin A$, then there is no $s \in S^x$ with $f^x(s) \geq (c_1, \ldots, c_k)$, and we return $\perp$. Otherwise there is some $s \in S^x_{\mathrm{opt}}$ with $f^x(s) = (c'_1, \ldots, c'_k) \geq (c_1, \ldots, c_k)$. We find $(c'_1, \ldots, c'_k)$ by a binary search using queries similar to x' and return $(c'_1, \ldots, c'_k)$.

2. $A \leq_{\mathrm{T}}^{\mathrm{p}} \mathrm{Val}(\text{D-}\mathcal{O})$: On input $\langle\langle x, \langle w\rangle\rangle, \langle z, y_k, \ldots, y_1\rangle'\rangle$, we reject if $z \neq \sum_{i=1}^{k} w_i y_i$. Otherwise we accept if and only if there is some $s \in S^x$ with $f^x(s) \geq (y_1, \ldots, y_k)$, which can be determined by a query to $\mathrm{Val}(\text{D-}\mathcal{O})$ on $\langle x, \langle y_1, \ldots, y_k\rangle\rangle$.

3. $\mathrm{Val}(\text{W-}\mathcal{O}) \leq_{\mathrm{T}}^{\mathrm{p}} g$: On input $\langle x, \langle w_1, \ldots, w_k\rangle\rangle$, we obtain $r := g(\langle x, \langle w_1, \ldots, w_k\rangle\rangle)$ by a query to the oracle. If $r = 0$, there are no $z, y_1, \ldots, y_k \in \mathbb{N}$ with $\langle\langle x, \langle w_1, \ldots, w_k\rangle\rangle, \langle z, y_k, \ldots, y_1\rangle'\rangle \in A$, and thus $S^x = \emptyset$ and we return $\perp$. Otherwise, let $z, y_1, \ldots, y_k \in \mathbb{N}$ with $\langle z, y_k, \ldots, y_1\rangle' = r$. Hence we have $z = \sum_{i=1}^{k} w_i y_i$ and $f^x(s) \geq (y_1, \ldots, y_k)$ for some $s \in S^x$.

   Assume there is some $s' \in S^x_{\mathrm{opt}}$ such that $z' := \sum_{i=1}^{k} w_i f_i^x(s') > \sum_{i=1}^{k} w_i f_i^x(s) \geq \sum_{i=1}^{k} w_i y_i$. Then $\langle z', f_k^x(s'), \ldots, f_1^x(s')\rangle' > \langle z, y_k, \ldots, y_1\rangle' = r$ because of the lexicographic ordering induced by $\langle\cdot\rangle'$ and thus r is not maximal, which is a contradiction.

   It remains to show that $(y_1, \ldots, y_k)$ is the value of some solution. Let s be the previously mentioned solution and assume that $f_i^x(s) > y_i$ for some i. Let $z' := \sum_{i=1}^{k} w_i f_i^x(s)$. If $z' > z$, then $\langle z', f_k^x(s), \ldots, f_1^x(s)\rangle' > r$, which is impossible. Otherwise $z' = z$ (and $w_i = 0$) and hence $\langle z', f_k^x(s), \ldots, f_1^x(s)\rangle' > r$, which is impossible again. Thus we have $f^x(s) = (y_1, \ldots, y_k)$, which is a valid answer for the input.

4. $g \leq_{\mathrm{T}}^{\mathrm{p}} \mathrm{Val}(\text{W-}\mathcal{O})$: On input $\langle x, \langle w_1, \ldots, w_k\rangle\rangle$, let $\tilde{w}_i := w_i \cdot 2^{k \cdot p(|x|)} + 2^{(i-1)\cdot p(|x|)}$ for all i and query $\mathrm{Val}(\text{W-}\mathcal{O})$ on $\langle x, \langle \tilde{w}_1, \ldots, \tilde{w}_k\rangle\rangle$. On answer $\perp$ we have $S^x = \emptyset$ and return 0, which is obviously the correct value. Otherwise, if $(y_1, \ldots, y_k)$ is the obtained answer, let the reduction function return $1 + \sum_{i=1}^{k} \tilde{w}_i y_i = 1 + \sum_{i=1}^{k} w_i y_i 2^{k \cdot p(|x|)} + \sum_{i=1}^{k} y_i 2^{(i-1)\cdot p(|x|)} = \langle z, y_k, \ldots, y_1\rangle'$ for $z = \sum_{i=1}^{k} w_i y_i$. Because we got $(y_1, \ldots, y_k)$ from a query to $\mathrm{Val}(\text{W-}\mathcal{O})$, there is some $s \in S^x$ such that $f^x(s) \geq (y_1, \ldots, y_k)$ and thus, the returned value is in $\mathrm{wit}_q \cdot A(\langle x, \langle w_1, \ldots, w_k\rangle\rangle)$. To see that it is indeed

maximal, assume there is some $\langle z', y'_1, \ldots, y'_k \rangle' \in \mathrm{wit}_q \cdot A(\langle x, \langle w_1, \ldots, w_k \rangle \rangle)$ that is strictly larger. Here we get

$$\begin{aligned}\sum_{i=1}^{k} \tilde{w}_i y'_i &= \sum_{i=1}^{k} w_i y'_i 2^{k \cdot p(|x|)} + \sum_{i=1}^{k} y'_i 2^{(i-1) \cdot p(|x|)} \\ &= \langle z', y'_k, \ldots, y'_1 \rangle' - 1 > \langle z, y_k, \ldots, y_1 \rangle' - 1 = \sum_{i=1}^{k} \tilde{w}_i y_i,\end{aligned}$$

which contradicts the fact that Val(W-$\mathcal{O}$) returns a value that is optimal with respect to the sum weighted by $(\tilde{w_1}, \ldots, \tilde{w_k})$. □

Corollary 5.16. Let $A, L, D, W \in \mathrm{NP}$. The following statements are equivalent:

1. There exists a multiobjective problem $\mathcal{O} = (S^x, f, \geq)$ such that

$$\begin{aligned} A &\equiv^{\mathrm{p}}_{\mathrm{T}} \mathrm{Val}(\text{A-}\mathcal{O}), \\ L &\equiv^{\mathrm{p}}_{\mathrm{T}} \mathrm{Val}(\text{L-}\mathcal{O}), \\ D &\equiv^{\mathrm{p}}_{\mathrm{T}} \mathrm{Val}(\text{D-}\mathcal{O}), \\ W &\equiv^{\mathrm{p}}_{\mathrm{T}} \mathrm{Val}(\text{W-}\mathcal{O}). \end{aligned}$$

2. $A \leq^{\mathrm{p}}_{\mathrm{T}} L \leq^{\mathrm{p}}_{\mathrm{T}} D, W$ and there is some $D' \in \mathrm{NP}$ and some $g \in \max \cdot D'$ such that $W \equiv^{\mathrm{p}}_{\mathrm{T}} g$ and $D' \equiv^{\mathrm{p}}_{\mathrm{T}} D$.

Proof. "2 ⇒ 1" follows from Theorem 5.13 applied to A, L, D', W and "1 ⇒ 2" follows from Proposition 5.15 and Theorem 5.1. □

Concerning the requirement "$W \equiv^{\mathrm{p}}_{\mathrm{T}} g$ for some $g \in \max \cdot D'$" in Corollary 5.16, observe that every set $X \in \mathrm{NP}$ is equivalent to some function $g \in \max \cdot Y$ for some $Y \equiv^{\mathrm{p}}_{\mathrm{T}} \textsc{Sat}$ (define $Y = \{\langle x, 3 + \mathrm{c}_X(x) \rangle \mid x \in \mathbb{N}\} \cup \{\langle x, 1 + \mathrm{c}_{\textsc{Sat}}(x) \rangle \mid x \in \mathbb{N}\}$). So for a problem $\mathcal{O}$ where Val(D-$\mathcal{O}$) is NP-hard, the complexity of Val(W-$\mathcal{O}$) can be arbitrary. The easier Val(D-$\mathcal{O}$) gets, the more restrictions are imposed on the complexity for Val(W-$\mathcal{O}$). However, this does not mean that Val(W-$\mathcal{O}$) needs to have lower complexity, since Val(W-$\mathcal{O}$) can be NP-hard while Val(D-$\mathcal{O}$) is polynomial-time solvable (take, for example, D as a witness set for Sat).

Corollary 5.17. If $A, L, W \in \mathrm{NP}$ such that $A \leq^{\mathrm{p}}_{\mathrm{T}} L \leq^{\mathrm{p}}_{\mathrm{T}} W$, then there exists a multiobjective problem $\mathcal{O}$ such that $A \equiv^{\mathrm{p}}_{\mathrm{T}} \mathrm{Val}(\text{A-}\mathcal{O})$, $L \equiv^{\mathrm{p}}_{\mathrm{T}} \mathrm{Val}(\text{L-}\mathcal{O})$, and $W \equiv^{\mathrm{p}}_{\mathrm{T}} \mathrm{Val}(\text{W-}\mathcal{O}) \equiv^{\mathrm{p}}_{\mathrm{T}} \mathrm{Val}(\text{D-}\mathcal{O})$.

Proof. Let $D = D' = \{\langle x, 1 \rangle \mid x \in W\}$ and $p(n) = 1$. Note that $D, D' \in \mathrm{NP}$, $D' \equiv^{\mathrm{p}}_{\mathrm{T}} D \equiv^{\mathrm{p}}_{\mathrm{T}} W$, and $\max_p \cdot D' \equiv^{\mathrm{p}}_{\mathrm{T}} W$. So we can apply Corollary 5.16, which finishes the proof. □

5.3 Complexities of Value Notions Individually

Having completely characterized the possible combinations of sets the value notions of a fixed multiobjective problem can be equivalent to, we now want to concentrate on

each value notion on its own. We will show that Val(L-$\mathcal{O}$), Val(D-$\mathcal{O}$), and Val(W-$\mathcal{O}$) are always equivalent to sets in NP. The converse was already shown in Corollary 5.16.

For Val(A-$\mathcal{O}$), the situation is different: We show that each function in NPMV is equivalent to some Val(A-$\mathcal{O}$). Together with Corollary 3.22 this implies that (under some assumption) there are problems $\mathcal{O}$ such that Val(A-$\mathcal{O}$) is inequivalent to any set and with Theorem 3.27, we get (under NEE $\neq$ coNEE) that there are problems $\mathcal{O}$ such that Val(A-$\mathcal{O}$) is inequivalent to any function in $\mathrm{coNPMV_t}$.

It is likely that the class $\{\text{Val(A-}\mathcal{O}) \mid \mathcal{O} \text{ is a multiobjective problem}\}$ is inequivalent to any known class of multivalued functions. Please see Figure 5.3 for an overview.

Corollary 5.18. For every multiobjective problem $\mathcal{O}$ the following holds.

1. Val(L-$\mathcal{O}$) $\equiv^{\mathrm{p}}_{\mathrm{T}} B$ for some $B \in \mathrm{NP}$.
2. Val(D-$\mathcal{O}$) $\equiv^{\mathrm{p}}_{\mathrm{T}} B$ for some $B \in \mathrm{NP}$.
3. Val(W-$\mathcal{O}$) $\equiv^{\mathrm{p}}_{\mathrm{T}} B$ for some $B \in \mathrm{NP}$.

Proof. By Proposition 4.5, we may assume that $\mathcal{O}$ is a k-objective problem $\mathcal{O} = (S, f, \geq)$.

1. Let $1, 2, \ldots, k$ be the order of objectives for Val(L-$\mathcal{O}$) and p be a polynomial upper bound for all values of f. Let

$$\begin{aligned} B = \{\langle x, \langle y_1, \ldots, y_k\rangle\rangle \mid x, y_1, \ldots, y_k \in \mathbb{N} \text{ and there is some } s \in S^x \text{ such that} \\ f_1(s) \geq y_1 \\ \wedge\, (f_1(s) = y_1 \implies (f_2(s) \geq y_2 \\ \wedge\, (f_2(s) = y_2 \implies (f_3(s) \geq y_3 \\ \ldots \\ \wedge\, (f_{k-1}(s) = y_{k-1} \implies f_k(s) \geq y_k) \ldots))))\} \end{aligned}$$

   and observe that $B \in \mathrm{NP}$. We have Val(L-$\mathcal{O}$) $\leq^{\mathrm{p}}_{\mathrm{T}} B$ by a binary search over k stages: suppose $(y^*_1, \ldots, y^*_k) \in \text{Val(L-}\mathcal{O})(x)$. In the i-th stage of the binary search, we ask queries of the form $\langle x, \langle y^*_1, \ldots, y^*_{i-1}, y_i, 0, \ldots, 0\rangle\rangle \in B$. This way we find y^*_i in polynomial time. On the other hand, given the value of Val(L-$\mathcal{O}$)(x), it is easy to determine whether or not $\langle x, \langle y_1, \ldots, y_k\rangle\rangle \in B$, hence we also have $B \leq^{\mathrm{p}}_{\mathrm{T}}$ Val(L-$\mathcal{O}$).

2. Follows from Proposition 5.15.

3. By Proposition 5.15, there exists a $g \in \max\cdot \mathrm{NP}$ such that Val(W-$\mathcal{O}$) $\equiv^{\mathrm{p}}_{\mathrm{T}} g$. By Proposition 3.11, $g \equiv^{\mathrm{p}}_{\mathrm{T}} B$ for some $B \in \mathrm{NP}$. □

Proposition 5.19. For any $g \in \mathrm{NPMV}$ there is some two-objective problem $\mathcal{O}$ such that $g \equiv^{\mathrm{p}}_{\mathrm{T}}$ Val(A-$\mathcal{O}$).

Proof. Let $g = \mathrm{wit}_p\cdot \exists_q\cdot B$ for some polynomials p, q and some $B \in \mathrm{P}$. By appropriately modifying B, we can assume that $p = q$ and for any $\langle\langle x, y\rangle, z\rangle \in B$ it holds that $y < 2^{p(|x|)}$ and $z < 2^{p(|\langle x,y\rangle|)}$. Define $\mathcal{O} = (S, f, \geq)$ where $S^x = \{\langle y, z\rangle \mid \langle\langle x, y\rangle, z\rangle \in B\}$ and $f^x(\langle y, z\rangle) = (y, 2^{p(|x|)} - y)$. Observe that $\mathcal{O}$ is a two-objective problem and that any value of f^x is optimal. Furthermore, for any $x, y \in \mathbb{N}$ it holds that $y \in g(x) \iff \exists s\, f^x(s) = (y, 2^{p(|x|)} - y)$. This means that $g \equiv^{\mathrm{p}}_{\mathrm{T}}$ Val(A-$\mathcal{O}$). □

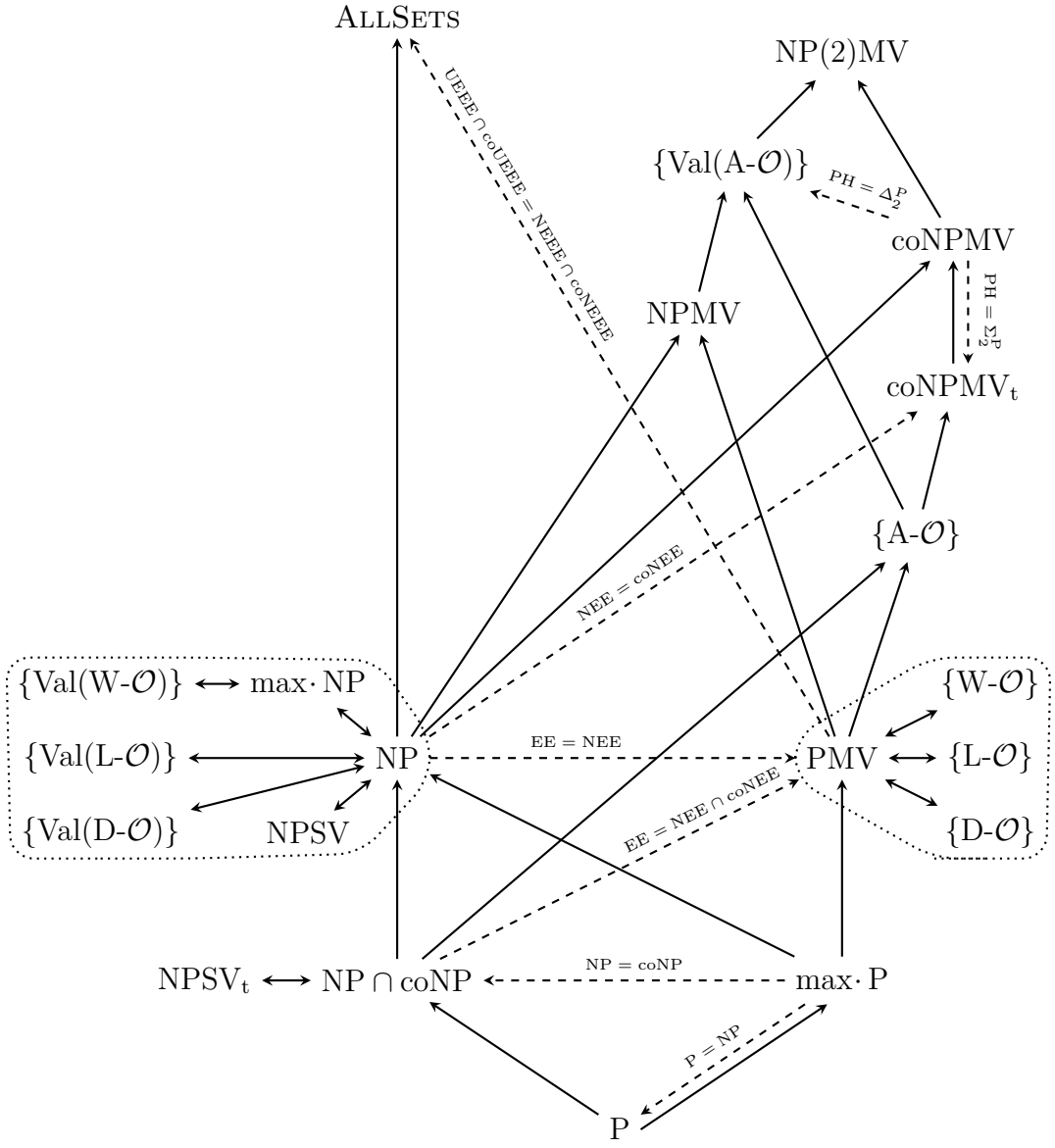


Figure 5.3: Extension of Figure 3.1 from page 43 by classes of search and value notions, where {X-$\mathcal{O}$} is shorthand for {X-$\mathcal{O}$ | $\mathcal{O}$ is a multiobjective problem} and similar for {Val(X-$\mathcal{O}$)}. A (non-dashed) arrow from $\mathcal{C}$ to $\mathcal{D}$ denotes that $\mathcal{C}$ can be embedded in $\mathcal{D}$, i. e. $\forall f \in \mathcal{C}\, \exists g \in \mathcal{D}\colon f \equiv_{\mathrm{T}}^{\mathrm{p}} g$. Dashed arrows give evidence against such an embedding (the embedding implies the arrow's label).
Observe that the embedding relation is reflexive and transitive and that evidence against an embedding propagates along non-dashed arrows (heads of dashed arrows can be moved downwards, tails can be moved upwards). This means that for any two classes of functions defined via multiobjective problems, we can either show that the first class can be embedded in the second class or we provide evidence against such an embedding.
Note that PMV = NPMV$_g$, max· NP = OptP (Krentel [Kre88]) and AllSets is the class of all decision problems.

Proposition 5.20. If PH $\neq \Delta_2^{\mathrm{P}}$, then there is some multiobjective problem $\mathcal{O}$ such that Val(A-$\mathcal{O}$) $\not\equiv_{\mathrm{T}}^{\mathrm{p}} g$ for any $g \in$ coNPMV.

Proof. We show the converse, i. e. if coNPMV $\subseteq_{\equiv_{\mathrm{T}}^{\mathrm{p}}}$ {Val(A-$\mathcal{O}$) | $\mathcal{O}$ is a multiobjective problem} then PH $= \Delta_2^{\mathrm{P}}$. By Theorem 5.1, we have Val(A-$\mathcal{O}$) $\leq_{\mathrm{T}}^{\mathrm{p}}$ Sat $\leq_{\mathrm{T}}^{\mathrm{p}}$ f-Sat for any multiobjective problem $\mathcal{O}$. Together with the assumption coNPMV $\subseteq_{\equiv_{\mathrm{T}}^{\mathrm{p}}}$ {Val(A-$\mathcal{O}$) | $\mathcal{O}$ is a multiobjective problem}, we obtain that any $g \in$ coNPMV reduces to f-Sat. Theorem 3.16 then shows that PH $= \Delta_2^{\mathrm{P}}$. □

5.4 Complexities of Search Notions Individually

We show that the search notions L-$\mathcal{O}$, D-$\mathcal{O}$, and W-$\mathcal{O}$ are equivalent to the functions in PMV, the class of NP search problems. This means that, opposed to the value notions from the previous section, the complexities of these search notions do not cover all problems in NP, unless EE = NEE (Theorem 3.25).

Similar to Val(A-$\mathcal{O}$), the search notion A-$\mathcal{O}$ again provides a greater range: With respect to embeddings, we show that the classes PMV and NP ∩ coNP provide lower bounds for A-$\mathcal{O}$ and the classes Val(A-$\mathcal{O}$) and coNPMV$_{\mathrm{t}}$ are upper bounds.

Again, the class {A-$\mathcal{O}$ | $\mathcal{O}$ is a multiobjective problem} seems to be inequivalent to known classes of multivalued functions. Please see Figure 5.3 on page 80 for an overview of the results obtained in this and the preceding section.

Theorem 5.21. Let $k \geq 1$ and $h = \mathrm{wit}_{\cdot} X$ for some $X \subseteq \mathbb{N}$. The following statements are equivalent:

1. There is some $g \in$ PMV such that $h \equiv_{\mathrm{T}}^{\mathrm{p}} g$.
2. There is some k-objective problem $\mathcal{O} = (S, f, \geq)$ such that $h \equiv_{\mathrm{T}}^{\mathrm{p}}$ L-$\mathcal{O}$.
3. There is some k-objective problem $\mathcal{O} = (S, f, \geq)$ such that $h \equiv_{\mathrm{T}}^{\mathrm{p}}$ D-$\mathcal{O}$.
4. There is some k-objective problem $\mathcal{O} = (S, f, \geq)$ such that $h \equiv_{\mathrm{T}}^{\mathrm{p}}$ W-$\mathcal{O}$.

Proof. "1 ⇒ 2, 3, 4": Define the k-objective problem $\mathcal{O} = (S, f, \geq)$ with $S^x = g(x)$ and $f^x(s) = (0, 0, \ldots, 0)$ for all $s \in S^x$. It holds that D-$\mathcal{O} \equiv_{\mathrm{T}}^{\mathrm{p}}$ W-$\mathcal{O} \equiv_{\mathrm{T}}^{\mathrm{p}}$ L-$\mathcal{O} = g \equiv_{\mathrm{T}}^{\mathrm{p}} h$.

"3 ⇒ 1": By Proposition 4.7, we know that D-$\mathcal{O} \in$ PMV, so we can set $g =$ D-$\mathcal{O}$.

"4 ⇒ 1": Note that by Proposition 4.6, W-$\mathcal{O}$ = A-$\mathcal{O}'$ for some single-objective problem $\mathcal{O}'$ and A-$\mathcal{O}' \equiv_{\mathrm{T}}^{\mathrm{p}}$ D-$\mathcal{O}'$ by Proposition 4.4 and we have just shown that in this case, there is some $g \in$ PMV such that $g \equiv_{\mathrm{T}}^{\mathrm{p}}$ D-$\mathcal{O}'$.

"2 ⇒ 1": This is analogous to the proof for "4 ⇒ 1". □

Proposition 5.22. For every $k \geq 1$ and every function $g \in$ PMV there is some k-objective problem $\mathcal{O}$ such that $g =$ A-$\mathcal{O}$.

Proof. Define the k-objective problem $\mathcal{O} = (S, f, \geq)$ with $S^x = g(x)$ and $f^x(s) = (0, 0, \ldots, 0)$ for all $s \in S^x$ and observe that $g(x) =$ A-$\mathcal{O}$. □

Theorem 5.23. For every $L \in$ NP ∩ coNP there is a two-objective problem $\mathcal{O}$ such that A-$\mathcal{O} \equiv_{\mathrm{T}}^{\mathrm{p}} L$.

Proof. Let $L \in \mathrm{NP} \cap \mathrm{coNP}$. Hence there are witness sets $L_1, L_2 \in \mathrm{P}$ and a polynomial p such that $L = \exists_p \cdot L_1$ and $\overline{L} = \exists_p \cdot L_2$, which means that

$$\begin{aligned} x \in L &\iff \exists y \text{ such that } y < 2^{p(|x|)} \text{ and } \langle x, y\rangle \in L_1 \\ x \notin L &\iff \exists y \text{ such that } y < 2^{p(|x|)} \text{ and } \langle x, y\rangle \in L_2 \end{aligned}$$

for all $x \in \mathbb{N}$. Note that L_1 and L_2 are disjoint. Let $\mathcal{O} = (S, f, \leq)$, where $S^x = (\mathrm{wit}_p \cdot L_1)(x) \cup (\mathrm{wit}_p \cdot L_2)(x) \cup \{2^{p(|x|)}, 2^{p(|x|)} + 1\}$ and

$$f^x(y) = \begin{cases} (1,0) & \text{if } y < 2^{p(|x|)} \text{ and } \langle x, y\rangle \in L_1 \\ (2,0) & \text{if } y = 2^{p(|x|)} \\ (0,1) & \text{if } y < 2^{p(|x|)} \text{ and } \langle x, y\rangle \in L_2 \\ (0,2) & \text{if } y = 2^{p(|x|)} + 1 \end{cases}$$

for all $x \in \mathbb{N}$ and $y \in S^x$. Observe that $\mathcal{O}$ is a 2-objective problem. We have the following reductions.

1. $L \leq_{\mathrm{T}}^{\mathrm{p}}$ A-$\mathcal{O}$: For all $x \in \mathbb{N}$ we have

$$\begin{aligned} x \in L &\iff \exists y \text{ such that } y < 2^{p(|x|)} \text{ and } \langle x, y\rangle \in L_1 \text{ and} \\ &\quad\ \forall y' \text{ with } y' < 2^{p(|x|)} \text{ we have } \langle x, y'\rangle \notin L_2 \\ &\iff \text{A-}\mathcal{O}(x) = (\mathrm{wit}_p \cdot L_2)(x) \cup \{2^{p(|x|)} + 1\} \end{aligned}$$

   and $x \notin L \iff \text{A-}\mathcal{O}(x) = (\mathrm{wit}_p \cdot L_2)(x) \cup \{2^{p(|x|)}\}$ analogously. If we get an arbitrary element from A-$\mathcal{O}(x)$ we can distinguish the two cases in polynomial time and thus $L \leq_{\mathrm{T}}^{\mathrm{p}}$ A-$\mathcal{O}$.

2. A-$\mathcal{O} \leq_{\mathrm{T}}^{\mathrm{p}} L$: For $x \in \mathbb{N}$, observe that $\{2^{p(|x|)}, 2^{p(|x|)} + 1\} \subseteq S^x$. We will argue that one of those solutions is optimal and, furthermore, this solution can be determined by a single query to L. For that purpose, observe that if $x \in L$, then for all $y < 2^{p(|x|)}$ we have $\langle x, y\rangle \notin L_2$, hence there is no y whose value dominates $(0, 2)$, and we can return $y = 2^{p(|x|)} + 1$ as solution for A-$\mathcal{O}(x)$. On the other hand, if $x \notin L$, then for all $y < 2^{p(|x|)}$ we have $\langle x, y\rangle \notin L_1$, hence there is no y whose value dominates $(2, 0)$, and we can return $y = 2^{p(|x|)}$ as solution for A-$\mathcal{O}(x)$. In all cases we compute a refinement of A-$\mathcal{O}$ and thus have A-$\mathcal{O} \leq_{\mathrm{T}}^{\mathrm{p}} L$ as claimed. □

Proposition 5.24. For every k-objective problem $\mathcal{O}$ there is some $f \in \mathrm{coNPMV_t}$ such that A-$\mathcal{O} \equiv_{\mathrm{T}}^{\mathrm{p}} f$.

Proof. Define the problem $\mathcal{O}'$ similar to $\mathcal{O}$ but for each instance x add a solution x_{bad} that is always present and is worse than all other solutions that can possibly exist. We thus have A-$\mathcal{O} \equiv_{\mathrm{T}}^{\mathrm{p}}$ A-$\mathcal{O}'$. The assertion follows since A-$\mathcal{O}'$ is total and lies in coNPMV by Proposition 4.7. □

Proposition 5.25. For every multiobjective problem $\mathcal{O} = (S, f, \geq)$ there is a multiobjective problem $\mathcal{O}' = (S, g, \geq)$ such that A-$\mathcal{O}$ = A-$\mathcal{O}' \equiv_{\mathrm{T}}^{\mathrm{p}} \mathrm{Val}$(A-$\mathcal{O}'$).

Proof. Let $\mathcal{O} = (S, f, \geq)$ be a k-objective problem and assume $k \geq 2$ (use the same objective function twice for $k = 1$). Let p be a polynomial such that for all x and all $s \in S^x$ it holds that $s < 2^{p(|x|)}$ and $f_i^x(s) < 2^{p(|x|)}$ for all $1 \leq i \leq k$. Define the k-objective problem $\mathcal{O}' = (S, g, \geq)$ where

$$g_i^x(s) = f_i^x(s)\, k\, 2^{3\,p(|x|)} + \sum_{j=1}^{k} f_j^x(s)\, 2^{p(|x|)} + \begin{cases} 2^{p(|x|)} - 1 - s & \text{for } i = 1 \\ s & \text{for } i \geq 2. \end{cases}$$

Claim 5.26. The following statements are equivalent for all $x \in \mathbb{N}$ and $s_1, s_2 \in S^x$:

1. $f^x(s_1) \neq f^x(s_2)$ and $f^x(s_1) \leq f^x(s_2)$
2. $g^x(s_1) \neq g^x(s_2)$ and $g^x(s_1) \leq g^x(s_2)$

Proof of the claim. "1 $\Rightarrow$ 2": Assume $f^x(s_1) \neq f^x(s_2)$ and $f^x(s_1) \leq f^x(s_2)$ and let $1 \leq j \leq k$ such that $f_j^x(s_1) < f_j^x(s_2)$. Since f_j occurs in each g_i with a factor of at least $2^{p(|x|)}$ and $s_1, s_2, 2^{p(|x|)} - 1 - s_1, 2^{p(|x|)} - 1 - s_2 < 2^{p(|x|)}$, we have $g_i^x(s_1) < g_i^x(s_2)$ for each i.

"2 $\Rightarrow$ 1": Assume $g^x(s_1) \neq g^x(s_2)$ and $g^x(s_1) \leq g^x(s_2)$. It is not possible that $f^x(s_1) = f^x(s_2)$, since in this case, $0 \neq s_1 - s_2 = g_1^x(s_2) - g_1^x(s_1) = -(g_2^x(s_2) - g_2^x(s_1))$, which contradicts the fact that $g^x(s_1) \leq g^x(s_2)$. Hence we have $f^x(s_1) \neq f^x(s_2)$. Finally, assume that there is some $1 \leq j \leq k$ such that $f_j^x(s_1) > f_j^x(s_2)$. Then we would also have $g_j^x(s_1) > g_j^x(s_2)$ because of the large factor $k\, 2^{3\,p(|x|)}$. ◇

From the claim it follows that a solution is not optimal in $\mathcal{O}$ if and only if it is not optimal in $\mathcal{O}'$ and thus the set of optimal solutions coincide, i. e. A-$\mathcal{O}$ = A-$\mathcal{O}'$. Furthermore, since the solution is encoded into the value for $\mathcal{O}'$, we obtain A-$\mathcal{O}' \equiv_{\mathrm{T}}^{\mathrm{p}}$ Val(A-$\mathcal{O}'$). □

Chapter 6

Approximation

If computing the exact solution for a problem is not possible in reasonable time, one often turns to computing approximate solutions. For many single-objective problems, there are algorithms that can generate solutions that are always guaranteed to be within a certain factor of the optimal solution. This concept of *approximation algorithms* can of course be extended to multiple objectives. Papadimitriou and Yannakakis [PY00] showed that every set of optimal solutions has a $(1+\varepsilon)$-approximation of size polynomial in the size of the instance and $1/\varepsilon$. Of course, it is not always clear that this set can also be obtained in polynomial time.

We continue our systematic study and define approximability notions for many of the solution notions from section 4.1 and investigate their relations. Figure 6.1 shows for arbitrary multiobjective problems in which cases polynomial time solvability of one such notions transfers to polynomial time solvability of another notion, and what quality of approximation can be preserved at least.

We reveal a significant dichotomy between the approximability of maximization and minimization problems. We show that if all objectives have to be minimized, then some approximability results translate from single-objective to multiobjective optimization. In contrast, such translations are not possible for problems where all objectives have to be maximized, unless P = NP.

6.1 Notions of Multiobjective Approximation

We discuss reasonable concepts of "approximately solving $\mathcal{O}$" for a k-objective problem $\mathcal{O} = (S, f, \leftarrow)$ where $\leftarrow$ is obtained from $\leftarrow_1, \ldots, \leftarrow_k$. Again, we use multivalued functions as models.

Definition 6.1. For approximations we need to relax the relation $\leftarrow$ by a factor of α (later called the *approximation ratio*). For any real $a \geq 1$ define $u \overset{a}{\leq} v \iff u \leq a \cdot v$ and $u \overset{a}{\geq} v \iff a \cdot u \geq v$. Let $p = (p_1, \ldots, p_k), q = (q_1, \ldots, q_k) \in \mathbb{N}^k$, and let $\alpha = (a_1, \ldots, a_k) \in \mathbb{R}^k$ where $a_1, \ldots, a_k \geq 1$. We say that p *weakly α-dominates* q, $p \overset{\alpha}{\leftarrow} q$ for short, if $p_i \overset{a_i}{\leftarrow}_i q_i$ for all $1 \leq i \leq k$. Again we extend $\overset{\alpha}{\leftarrow}$ to combinations of values and solutions, if f and x are clear from the context.

In order to examine minimization, maximization and mixed problems at the same time, we always regard approximation ratios to be larger than one. Sometimes, approximation

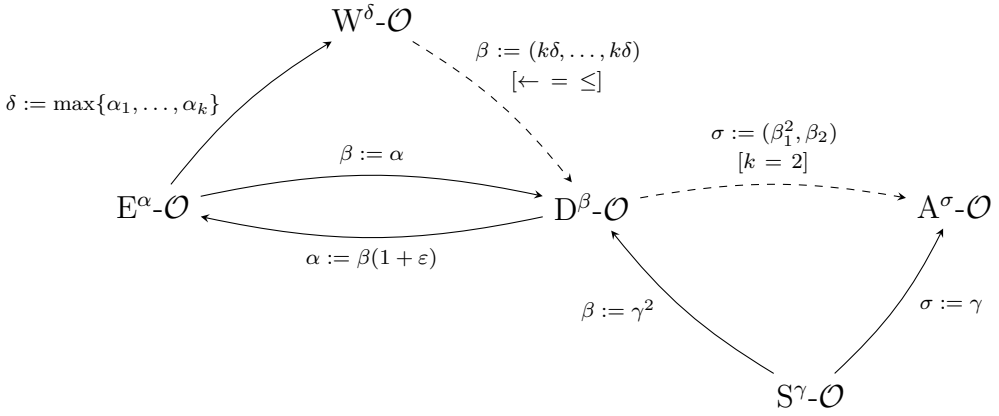


Figure 6.1: Implications between polynomial time solvability of approximate solution notions for any k-objective problem $\mathcal{O} = (S, f, \leftarrow)$ where $\varepsilon > 0$ can be chosen arbitrarily close to zero. For instance, it holds that if D^{β}-$\mathcal{O}$ is polynomial-time solvable then for each $\varepsilon > 0$, also $\mathrm{E}^{\beta(1+\varepsilon)}$-$\mathcal{O}$ is polynomial-time solvable. Dashed lines indicate a conditional implication where the condition is shown in brackets. Note that $\alpha, \beta, \gamma, \sigma \in \mathbb{R}^k$, $\delta \in \mathbb{R}$ and W^{δ}-$\mathcal{O}$ is only defined if all objectives are minimized or all are maximized.

ratios for maximization problems are considered to be between zero and one, and indeed we will do this in section 7.2. One can obtain this view by simply inverting the ratios here.

Note that for all $p, q, r \in \mathbb{N}^k$ it holds that $p \overset{\alpha}{\leftarrow} p$, and $p \overset{\alpha}{\leftarrow} q \overset{\beta}{\leftarrow} r \implies p \overset{\alpha\cdot\beta}{\leftarrow} r$, where $\alpha \cdot \beta$ is the component-wise multiplication. In particular, from $p \overset{\alpha}{\leftarrow} q$ and $q \leftarrow r$ it follows that $p \overset{\alpha}{\leftarrow} r$ and thus p weakly α-dominates all solutions that are dominated by q.

We already noted that computing the complete set of optimal solutions is not an appropriate solution notion since this set is typically exponentially large. In contrast, approximations of the complete set of Pareto-optimal points are very useful and feasible. Papadimitriou and Yannakakis [PY00] show that each set of Pareto-optimal solutions of a multiobjective problem $\mathcal{O} = (S, f, \leftarrow)$ has a polynomially-sized approximation. To clarify what this means, note that an ε-approximation of a set of single-objective solutions S^x can be seen as a single solution $s' \in S^x$ such that for all $s \in S^x$ it holds that $s' \overset{\varepsilon}{\leftarrow} s$. The obvious extension of this to multiple objectives is the following: A set $S' \subseteq S^x$ is called an *α-approximation* of S^x if for all $s \in S^x$ there is some $s' \in S'$ such that $s' \overset{\alpha}{\leftarrow} s$. The result by Papadimitriou and Yannakakis now states that for all α there exists a polynomial p such that for all x there exists an α-approximation S' of S^x with $\#S' \leq p(|x|)$. Hence there are small and quite precise approximations for S^x_{opt}, but in many cases it is hard to compute these approximations.

Since the approximate solution notion E^{α}-$\mathcal{O}$ exactly captures the task of computing such approximations, we will consider this solution notion when giving approximation algorithms for specific problems in chapter 7. Furthermore, we will call a problem $\mathcal{O}$ *α-approximable* if E^{α}-$\mathcal{O}$ is polynomial-time solvable. The other approximate solution notions are extensions of the solution notions from section 4.1 to approximation. Observe that each of these notions coincides with its exact version if $\alpha = (1, \ldots, 1)$ or $\delta = 1$, respectively.

Definition 6.2. Let $\mathcal{O} = (S, f, \leftarrow)$ be a k-objective problem and $\alpha = (\alpha_1, \ldots, \alpha_k) \in \mathbb{R}^k$ such that $\alpha_i \geq 1$ for all i and let $\delta \in \mathbb{R}$ such that $\delta \geq 1$.

E^α-$\mathcal{O}$ α-Approximate Every-Solution Notion
Compute a set of solutions that α-dominates every solution.
Input: instance x
Function: $\mathrm{E}^\alpha\text{-}\mathcal{O}(x) = \{S' \subseteq S^x \mid \forall s \in S^x\, \exists s' \in S' \colon s' \overset{\alpha}{\leftarrow} s\}$

A^α-$\mathcal{O}$ α-Approximate Arbitrary-Optimum Notion
Compute a solution that weakly α-dominates an arbitrary optimal solution.
Input: instance x
Function: $\mathrm{A}^\alpha\text{-}\mathcal{O}(x) = \{s \in S^x \mid \exists t \in S^x_{\mathrm{opt}} \colon s \overset{\alpha}{\leftarrow} t\}$

S^α-$\mathcal{O}$ α-Approximate Specific-Optimum Notion
Compute a solution that weakly α-dominates an optimal solution specified by a given cost vector.
Input: instance x, cost vector $c \in \mathbb{N}^k$
Function: $\mathrm{S}^\alpha\text{-}\mathcal{O}(\langle x, \langle c\rangle\rangle) = \{s \in S^x \mid \exists t \in S^x_{\mathrm{opt}} \colon s \overset{\alpha}{\leftarrow} t \overset{\alpha}{\leftarrow} c\} \cup \{\bot \mid \forall s \in S^x \colon s \not\leftarrow c\}$

D^α-$\mathcal{O}$ α-Approximate Dominating-Solution Notion
Compute a solution that weakly α-dominates a given cost vector.
Input: instance x, cost vector $c \in \mathbb{N}^k$
Function: $\mathrm{D}^\alpha\text{-}\mathcal{O}(\langle x, \langle c\rangle\rangle) = \{s \in S^x \mid s \overset{\alpha}{\leftarrow} c\} \cup \{\bot \mid \forall s \in S^x \colon s \not\leftarrow c\}$

W^δ-$\mathcal{O}$ δ-Approximate Weighted-Sum Notion (if all objectives are minimized or all maximized)
Single-objective problem that weights the objectives in a given way.
Input: instance x, weight vector $\omega \in \mathbb{N}^k$
Function: $\mathrm{W}^\delta\text{-}\mathcal{O}(\langle x, \langle \omega\rangle\rangle) = \{s \in S^x \mid \omega^{\mathrm{T}} \cdot f^x(s) \overset{\delta}{\leftarrow}_1 \omega^{\mathrm{T}} \cdot f^x(s') \text{ for all } s' \in S^x\}$

$\mathrm{C}^\delta_{\mathrm{i}}$-$\mathcal{O}$ δ-Approximate Constraint Notion for the i-th Objective
Single-objective problem that approximates the i-th objective while respecting constraints on the remaining objectives.
Input: instance x, constraint vector $b \in \mathbb{N}^k$ (the i-th component is ignored)
Function: $\mathrm{C}^\delta_{\mathrm{i}}\text{-}\mathcal{O}(x) = \{x \in S^x \mid x \in S^x_{\mathrm{con}} \text{ and } s \overset{\delta}{\leftarrow}_i \mathrm{opt}_i(S^x_{\mathrm{con}}) \text{ for } S^x_{\mathrm{con}} := \{s \in S^x \mid \forall j \neq i \colon f^x_j(s) \leftarrow_j b_j\}\}$

The vector α is called *approximation ratio*.

Note that we use the special symbol $\bot$ in the definition of S^α-$\mathcal{O}$ and D^α-$\mathcal{O}$ and assume that it is encoded as a natural number different from all elements in S^x. This symbol also occurs in the definition of the polynomial-time Turing-reduction between multivalued functions to represent the value "undefined". Since the functions S^α-$\mathcal{O}$ and D^α-$\mathcal{O}$ are always total, a confusion with this meaning cannot occur. Furthermore, note that the unions in the definitions are not necessarily disjoint. The approximate solution notion D^α-$\mathcal{O}$ corresponds to the gap problem used by Papadimitriou and Yannakakis [PY00]. We will get back to this in Theorem 6.3.

The performance of approximations where the optimized value is a single number like W-$\mathcal{O}$ is specified by a real number δ instead of a vector of real numbers α. Similar to W-$\mathcal{O}$, we consider W^δ-$\mathcal{O}$ only for multiobjective problems where all objectives have to be

minimized (resp., maximized). Note that in this case, $\leftarrow_1$ above can be replaced by any of the $\leftarrow_i$.

We disregard approximations for the lexicographical problem L-$\mathcal{O}$, since here it is not clear how to measure the performance of the approximation. Note that if $\mathcal{O}$ is a single-objective problem, then A^{δ}-$\mathcal{O}$, S^{δ}-$\mathcal{O}$, D^{δ}-$\mathcal{O}$, W^{δ}-$\mathcal{O}$, and C_1^{δ}-$\mathcal{O}$ are polynomial-time Turing equivalent to computing a δ-approximation for the single-objective problem.

Further research. Although D-$\mathcal{O}$ and $\mathrm{C_i}$-$\mathcal{O}$ are polynomial-time Turing equivalent, they can differ considerably with respect to their approximability. For example, for $\mathcal{O}$ = 2-EDC (2-objective exact disk cover, [GRS08]) and for every $\delta > 1$ it holds that $\mathrm{E}^{(\delta,\delta)}$-$\mathcal{O}$ and $\mathrm{D}^{(\delta,\delta)}$-$\mathcal{O}$ are polynomial-time solvable. In contrast, for every $\delta > 1$ it holds that $\mathrm{C}_1^{(\delta,\delta)}$-$\mathcal{O}$ and $\mathrm{C}_2^{(\delta,\delta)}$-$\mathcal{O}$ are not polynomial-time solvable, unless P = NP [GRS08].

Vassilvitskii and Yannakakis [VY05] investigate the problem of computing a good approximation for S^x_{opt} using as few solutions as possible. Here for a given instance x and a maximum number of solutions l, one has to find some $S \subseteq S^x$ with $\#S \leq l$ such that (1) there is some $\alpha \geq 1$ such that S is a solution to E^{α}-$\mathcal{O}$ on input x, and (2) there are no $\alpha' < \alpha$ and $S' \subseteq S^x$ with $\#S' \leq l$ such that S' is a solution to $\mathrm{E}^{\alpha'}$-$\mathcal{O}$ on input x.

For k-objective problems the performance of an approximation is given by a vector $\alpha \in \mathbb{R}^k$. Two such vectors can be incomparable and hence trade-offs are possible at the level of approximability. A typical 2-objective problem does not have *a single best* approximation ratio α, but there may exist a trade-off curve of incomparable best approximation ratios. In section 7.1, we discuss evidence suggesting that the traveling salesperson problem has such approximation trade-offs.

6.2 Relations Between the Approximation Notions

We study relationships among the approximate solution notions defined above. Papadimitriou and Yannakakis [PY00] demonstrate a close connection between E-$\mathcal{O}$ and D-$\mathcal{O}$ in the sense that up to a factor of $1 + \varepsilon$ both notions have the same approximability. We restate this result in a slightly extended version.

Theorem 6.3. Let $\mathcal{O} = (S, f, \leftarrow)$ be a k-objective problem, $\alpha = (a_1, \ldots, a_k)$ where $a_i \geq 1$, and $\varepsilon = (\varepsilon_1, \ldots, \varepsilon_k)$ where $\varepsilon_i > 0$.

1. E^{α}-$\mathcal{O}$ polynomial-time solvable $\implies$ D^{α}-$\mathcal{O}$ polynomial-time solvable
2. $\mathrm{E}^{\alpha(1+\varepsilon)}$-$\mathcal{O}$ $\leq_{\mathrm{T}}^{\mathrm{p}}$ D^{α}-$\mathcal{O}$, where the running time of the reduction machine is polynomially bounded even in $|x| + \sum_i {}^1\!/_{\varepsilon_i}$

Proof. Let $\mathcal{O} = (S, f, \leftarrow)$ be some k-objective problem.

1. Suppose we obtain S' as a solution to E^{α}-$\mathcal{O}$ in polynomial time. To solve D^{α}-$\mathcal{O}$ for some $c \in \mathbb{N}^k$ in polynomial time, we check whether there is some $s' \in S'$ with $s' \overset{\alpha}{\leftarrow} c$. If there is such $s' \in S'$, we are done. On the other hand, if there is no $s' \in S'$ with $s' \overset{\alpha}{\leftarrow} c$, there cannot be any $s \in S^x$ with $s \leftarrow c$ (otherwise S' is not a valid solution to E^{α}-$\mathcal{O}$, because there is no solution $s' \in S'$ that α-dominates s), so we can return $\perp$.

2. We will construct some polynomial-sized lattice that covers the entire solution value space and then show how to find approximations of arbitrary solutions.

   By the definition of k-objective problems, there is some polynomial p such that $f_i^x(s) \leq 2^{p(|x|)}$ for every instance x, every solution $s \in S^x$, and $1 \leq i \leq k$.

   Fix some problem instance x, and let $\delta = \min_{1 \leq i \leq k} \varepsilon_i$, $r = \lceil \frac{1}{\delta} \rceil$, and $t = r \cdot p(|x|) + 1$. We show that $(1+\delta)^t$ bounds the value of every objective. From

   $$(1+\delta)^r = \sum_{j=0}^{r} \binom{r}{j} \delta^j \geq \binom{r}{0}\delta^0 + \binom{r}{1}\delta^1 \geq 1 + r \cdot \delta \geq 1 + \frac{1}{\delta} \cdot \delta = 2$$

   we obtain

   $$f_i^x(s) \leq 2^{p(|x|)} < 2^{p(|x|)}(1+\delta) \leq ((1+\delta)^r)^{p(|x|)}(1+\delta) = (1+\delta)^{r \cdot p(|x|)+1} = (1+\delta)^t.$$

   Next, consider the sets $I = \{(1+\delta)^j \mid 0 \leq j \leq t\}$ and $L = (I \cup \{0\})^k$. The elements of L cover the entire solution value space in the sense that for every solution $s \in S^x$ there is some $l \in L$ such that $l \overset{(1+\delta)}{\leftarrow} s \leftarrow l$. It even holds that $\lfloor l \rceil \overset{(1+\delta)}{\leftarrow} s \leftarrow \lfloor l \rceil$, where $\lfloor l \rceil$ denotes the vector obtained from l by rounding each component towards its direction of optimization. By $s \leftarrow \lfloor l \rceil$, given x and $\lfloor l \rceil$ as input, any solution for D^α-$\mathcal{O}$ must consist of some $s' \in S^x$ with $s' \overset{\alpha}{\leftarrow} \lfloor l \rceil$, thus $s' \overset{\alpha(1+\delta)}{\leftarrow} s$.

   To solve $\mathrm{E}^{\alpha(1+\varepsilon)}$-$\mathcal{O}$, it hence suffices to solve D^α-$\mathcal{O}$ for every $\lfloor l \rceil$ where $l \in L$ (note that $\delta \leq \varepsilon_i$ for all i). This is possible in time polynomial in $|x| + \sum_{i=1}^{k} \frac{1}{\varepsilon_i}$ using queries to D^α-$\mathcal{O}$, since $\#L = (\lceil \frac{1}{\delta} \rceil \cdot p(|x|) + 3)^k$. □

Note that the algorithm for $\mathrm{E}^{\alpha(1+\varepsilon)}$-$\mathcal{O}$ calls D^α-$\mathcal{O}$ at every point of some polynomial-sized lattice built over ε. If, however, $\mathcal{O} = (S, f, \leftarrow)$ is polynomially bounded (i. e. the values of f are polynomially bounded), we can instead call D^α-$\mathcal{O}$ for every possible solution value and thereby obtain a solution to E^α-$\mathcal{O}$. Hence, for multiobjective problems that are polynomially bounded, E^α-$\mathcal{O}$ and D^α-$\mathcal{O}$ are equivalent.

Theorem 6.4. Let $\mathcal{O} = (S, f, \leftarrow)$ be a k-objective problem and $\alpha = (\alpha_1, \ldots, \alpha_k)$ with $a_i \geq 1$.

1. D^{α^2}-$\mathcal{O} \leq_{\mathrm{T}}^{\mathrm{p}} \mathrm{S}^\alpha$-$\mathcal{O}$ and A^α-$\mathcal{O} \leq_{\mathrm{T}}^{\mathrm{p}} \mathrm{S}^\alpha$-$\mathcal{O}$
2. $\mathrm{A}^{(\alpha_1^2, \alpha_2)}$-$\mathcal{O} \leq_{\mathrm{T}}^{\mathrm{p}} \mathrm{D}^\alpha$-$\mathcal{O}$ if $k = 2$
3. If all objectives have to be minimized (resp., maximized) and E^α-$\mathcal{O}$ is polynomial-time solvable, then $\mathrm{W}^{\max_i(\alpha_i)}$-$\mathcal{O}$ is polynomial-time solvable.

Proof. Let $\mathcal{O} = (S, f, \leftarrow)$ be a k-objective problem with $\leftarrow = (\leftarrow_1, \ldots, \leftarrow_k)$, instance x and $\alpha = (a_1, \ldots, a_k)$ with $a_i \geq 1$.

1. We call S^α-$\mathcal{O}$ for instance x and cost vector c as they are given in the input for D^α-$\mathcal{O}$. If S^α-$\mathcal{O}$ returns $\perp$ (i. e. there is no $s \in S^x$ such that $s \leftarrow c$), we return $\perp$. Otherwise, S^α-$\mathcal{O}$ returns some $s \in S^x$, hence there is some $t \in S^x_{\mathrm{opt}}$ with $s \overset{\alpha}{\leftarrow} t \overset{\alpha}{\leftarrow} c$, which implies $s \overset{\alpha^2}{\leftarrow} c$, hence s is a solution to D^{α^2}-$\mathcal{O}$.

For the second part, we call S^α-$\mathcal{O}$ with some cost vector c such that c is dominated by the entire solution value space. If $S^x \neq \emptyset$, we get some $s \in S^x$ such that $s \overset{\alpha}{\leftarrow} t \overset{\alpha}{\leftarrow} c$ for some $t \in S^x_{\mathrm{opt}}$, which solves A^α-$\mathcal{O}$.

2. Suppose $k = 2$, $S^x \neq \emptyset$, and let $s \in \mathrm{opt}_2(\mathrm{opt}_1(S^x))$. Clearly, s is optimal. Given an oracle for D^α-$\mathcal{O}$, we perform a binary search over S^x that optimizes f^x_1 and obtain a solution $s' \in S^x$ with $f^x_1(s') \overset{\alpha_1}{\leftarrow}_1 f^x_1(s)$. However, s' might have an inappropriate value in f^x_2. For that reason, we minimize f^x_2 through a second binary search where we call D^α-$\mathcal{O}$ again and keep the first component of the cost vector fixed to $f^x_1(s')$. Since $f^x(s) \leftarrow (f^x_1(s'), f^x_2(s))$, this binary search finds a solution $\hat{s} \in S^x$ with $\hat{s} \overset{\alpha}{\leftarrow} (f^x_1(s'), f^x_2(s))$, and together with $f^x_1(s') \overset{\alpha_1}{\leftarrow}_1 f^x_1(s)$ we get $f^x_1(\hat{s}) \overset{\alpha^2_1}{\leftarrow}_1 f^x_1(s)$ and $f^x_2(\hat{s}) \overset{\alpha_2}{\leftarrow}_2 f^x_2(s)$.

3. Suppose all objectives have to be minimized (the theorem can be shown analogously if all objectives have to be maximized). For any instance x and weight vector $\omega = (\omega_1, \omega_2, \ldots, \omega_k)$, if $S^x \neq \emptyset$ then there is some $\hat{s} \in S^x$ that minimizes $\sum_{i=1}^k \omega_i f^x_i$. Let S' be a solution of E^α-$\mathcal{O}$. Then there must be a solution $s \in S'$ such that $s \overset{\alpha}{\leftarrow} \hat{s}$, hence $f^x_i(s) \leq \alpha_i f^x_i(\hat{s})$ for all i, which implies

$$\sum_{i=1}^k \omega_i f^x_i(s) \leq \sum_{i=1}^k \omega_i \alpha_i f^x_i(\hat{s}) \leq \max_i(\alpha_i) \sum_{i=1}^k \omega_i f^x_i(\hat{s}) \leq \max_i(\alpha_i) \sum_{i=1}^k \omega_i f^x_i(s')$$

for all $s' \in S^x$. It hence suffices to return a solution $s^* \in S'$ that minimizes $\sum_{i=1}^k \omega_i f^x_i$, which can be extracted from S' in polynomial time, because S' has polynomial cardinality. □

Further research. Even though for a fixed problem $\mathcal{O}$, the search notion A-$\mathcal{O}$ can be reduced to all other exact search notions for $\mathcal{O}$, it is not clear whether similar implications hold for A^α-$\mathcal{O}$. For $k > 2$, the reducibility of A^α-$\mathcal{O}$ to approximate problem notions other than S^α-$\mathcal{O}$ remains open. Moreover, we are interested in improvements or lower bounds for the implications in Figure 6.1 on page 86.

6.3 Pareto- versus Scalar Minimization

We show that for multiobjective problems $\mathcal{O}$ where all objectives have to be minimized, approximability results translate from single-objective to multiobjective optimization, especially when the problems are linear. In particular, if W^δ-$\mathcal{O}$ is polynomial-time solvable (which coincides with the single-objective problem for linear problems), then there exists some $c \geq 1$ such that $\mathrm{D}^{c\delta}$-$\mathcal{O}$ and $\mathrm{E}^{c\delta}$-$\mathcal{O}$ are polynomial-time solvable. Note that we show in the next section that surprisingly, this result does not hold for maximization problems (unless P = NP).

We will also generalize W^δ-$\mathcal{O}$ from weighted sums to general weighted norms and observe that D^δ-$\mathcal{O}$ corresponds to the notion for weighted maximum norms, whereas the ordinary W^δ-$\mathcal{O}$ is the weighted 1-norm.

Let us first review some properties of norms as they are important in this section.

Definition 6.5. A norm $\|\cdot\|$ on $\mathbb{R}^k$ is *monotone*, if for all vectors $x = (x_1, \ldots, x_k)^T, y = (y_1, \ldots, y_k)^T \in \mathbb{R}^k$ it holds that

$$|x_1| \leq |y_1| \wedge \cdots \wedge |x_k| \leq |y_k| \Rightarrow \|x\| \leq \|v\|.$$

Two norms $\|\cdot\|_a$ and $\|\cdot\|_b$ on the same space are *equivalent*, if there exist constants $c_1, c_2 > 0$ such that for all x,

$$c_1\|x\|_b \leq \|x\|_a \leq c_2\|x\|_b.$$

It is well known that all norms on $\mathbb{R}^k$ are equivalent. Important examples of norms on $\mathbb{R}^k$ are the *p-norms* $\|(x_1, \ldots, x_k)^T\|_p = (\sum_{i=1}^k |x_i|^p)^{1/p}$ defined for real numbers $p \geq 1$ and the *maximum norm* $\|(x_1, \ldots, x_k)^T\|_\infty = \max_i |x_i|$, which are all monotone, even if the components of the vectors are weighted by fixed non-negative numbers. Furthermore, for any $p \geq 1$ and any $x \in \mathbb{R}^k$ it holds that $\|x\|_\infty \leq \|x\|_p \leq k^{1/p}\|x\|_\infty$.

The next two lemmas tell us how to translate approximations for weighted norms of vectors to the weak approximate dominance relation and vice-versa.

Lemma 6.6. Let $k \geq 1$ and let $\leftarrow = (\leq, \ldots, \leq)$ be the k-dimensional $\leq$. For any norm $\|\cdot\|$ on $\mathbb{R}^k$ there is some $\hat{c} \geq 1$ such that for any $\delta \geq 1$ and $x, v \in \mathbb{N}^k$

$$x \overset{(\delta,\ldots,\delta)}{\leftarrow} v \qquad \Longrightarrow \qquad \|x\| \overset{\hat{c}\delta}{\leq} \|v\|.$$

In particular, if $\|\cdot\|$ is monotone then $\hat{c} = 1$, which is the case for any (weighted) p-norm and the (weighted) maximum norm.

Proof. First, suppose $\|\cdot\|$ is monotone. We get

$$\begin{aligned} x \overset{(\delta,\ldots,\delta)}{\leftarrow} v &\Longrightarrow (x_1 \leq \delta v_1 \wedge \cdots \wedge x_k \leq \delta v_k) \\ &\Longrightarrow \|x\| \leq \|(\delta v_1, \ldots, \delta v_k)^T\| \\ &\Longrightarrow \|x\| \leq \delta\|v\|, \end{aligned}$$

which shows the lemma for the monotone case. Next, suppose $\|\cdot\|$ is not monotone. By the equivalence of norms on $\mathbb{R}^k$ there are constants $c_1, c_2 > 0$ such that $x \overset{(\delta,\ldots,\delta)}{\leftarrow} v$ implies

$$c_1\|x\| \leq \|x\|_\infty \leq \delta\|v\|_\infty \leq c_2\delta\|v\|,$$

which yields the desired result for the general case with $\hat{c} = \frac{c_2}{c_1}$. Furthermore, observe that $c_1\|x\| \leq \|x\|_\infty \leq c_2\|x\|$ immediately implies $\frac{c_2}{c_1} \geq 1$. □

Lemma 6.7. Let $k \geq 1$ and let $\leftarrow = (\leq, \ldots, \leq)$ be the k-dimensional $\leq$. For any norm $\|\cdot\|$ on $\mathbb{R}^k$ there is some $\tilde{c} \geq 1$ such that for all $\delta \geq 1$ and $x, v \in \mathbb{N}^k$

$$\|Wx\| \leq \delta\|Wv\| \qquad \Longrightarrow \qquad x \overset{(\tilde{c}\delta,\ldots,\tilde{c}\delta)}{\leftarrow} v$$

where $W = \mathrm{diag}(\omega_1, \ldots, \omega_k)$ and

$$\omega_i = \begin{cases} \lceil \delta\tilde{c} \rceil + 1 & \text{if } v_i = 0 \\ 1/v_i & \text{if } v_i \neq 0. \end{cases}$$

In particular, if $\|\cdot\|$ is a p-norm then $\tilde{c} = k^{1/p}$ and $\tilde{c} = 1$ for the maximum norm.

Proof. By the equivalence of norms on $\mathbb{R}^k$, there is some $C > 0$ such that $\|x\|_\infty \leq C\|x\|$ for all $x \in \mathbb{R}^k$. Set $\tilde{c} = C \max_{b\in\{0,1\}^k} \|b\|$ ($b = (1,\dots,1)^T$ if the norm is monotone) and observe that the following inequality holds for any i, any $x \in \mathbb{N}^k$ and any $v \in \mathbb{N}^k$.

$$\omega_i x_i = (Wx)_i \leq \|Wx\|_\infty \leq C\|Wx\| \leq C\delta\|Wv\| \overset{(*)}{\leq} C\delta \max_{b\in\{0,1\}^k} \|b\| = \delta\tilde{c}$$

Note that for the inequality $(*)$, we used that $Wv \in \{0,1\}^k$.

If $v_i \neq 0$, this means that $x_i \leq \tilde{c}\delta v_i$ and for $v_i = 0$, we get $x_i \leq \frac{\delta\tilde{c}}{\lceil\delta\tilde{c}\rceil+1} < 1$ and thus $x_i = 0$ (since $x_i \in \mathbb{N}$), which again yields $0 = x_i \leq \tilde{c}\delta v_i = 0$. Thus, we get $x_i \leq \tilde{c}\delta v_i$ for any i and the main part of the lemma is proved.

The second part is obtained by observing that $C = 1$ for any p-norm and for the maximum norm and that $\|(1,\dots,1)^T\|_p = k^{1/p}$ for any p-norm and $\|(1,\dots,1)^T\|_\infty = 1$. □

In order to apply these results to multiobjective optimization, we need to generalize the definition of the weighted-sum notion to a weighted-norm notion.

Definition 6.8. For some k-objective problem $\mathcal{O} = (S, f, \leq)$, some norm $\|\cdot\|$ on $\mathbb{R}^k$, and some $\delta \geq 1$ we define the following.

$\mathrm{W}^\delta_{\|\cdot\|}$-$\mathcal{O}$ **δ-Approximate Weighted-Norm Notion**
Single-objective problem that first weights the objectives and then applies a norm.
Input: instance x, weight vector $\omega \in \mathbb{N}^k$
Function: $\mathrm{W}^\delta_{\|\cdot\|}$-$\mathcal{O}(\langle x, \langle\omega\rangle\rangle) = \{s \in S^x \mid \|Wf^x(s)\| \leq \delta\|Wf^x(s')\|$ for all $s' \in S^x$ where $W = \mathrm{diag}(\omega_1,\dots,\omega_k)\}$

This notion generalizes W^δ-$\mathcal{O}$, since W^δ-$\mathcal{O} = \mathrm{W}^\delta_{\|\cdot\|_1}$-$\mathcal{O}$, the δ-approximate weighted-1-norm notion. Note that the above definition can easily be extended for problems where all objectives have to be maximized.

Proposition 6.9. For any norm $\|\cdot\|$ on $\mathbb{R}^k$ there is some $c \geq 1$ such that for any k-objective problem $\mathcal{O} = (S, f, \leq)$ and any $\delta \geq 1$ it holds that

$$\mathrm{D}^{(c\delta,\dots,c\delta)}\text{-}\mathcal{O} \leq^{\mathrm{p}}_{\mathrm{T}} \mathrm{W}^\delta_{\|\cdot\|}\text{-}\mathcal{O}.$$

In particular, if $\|\cdot\|$ is a p-norm then $c = k^{1/p}$ and $c = 1$ for the maximum norm.

Proof. We show how $\mathrm{D}^{(c\delta,\dots,c\delta)}$-$\mathcal{O}$ can be solved in polynomial time relative to $\mathrm{W}^\delta_{\|\cdot\|}$-$\mathcal{O}$. Let the instance $x \in \mathbb{N}$ and the cost vector $v \in \mathbb{N}^k$ be the input. For the sake of clarity, we use $\leftarrow$ instead of $\leq$ in a multidimensional context. Let $\hat{c}, \tilde{c} \geq 1$ be the constants from Lemmas 6.6 and 6.7 corresponding to $\|\cdot\|$ and let $c = \hat{c}\tilde{c}$. Furthermore, let (as in Lemma 6.7) $\omega_i = \lceil\delta\tilde{c}\rceil + 1$ if $v_i = 0$ and $\omega_i = 1/v_i$ if $v_i \neq 0$ and let V be the product of all nonzero entries in v ($V = 1$ if $v = (0,\dots,0)$) and $\omega'_i = V\omega_i$. Note that the weights ω'_i are natural numbers, so we can call the algorithm for $\mathrm{W}^\delta_{\|\cdot\|}$-$\mathcal{O}$ with weights ω'_i. If we get a solution s from this call, we return s if $f^x(s) \overset{(c\delta,\dots,c\delta)}{\leftarrow} v$. In all other cases, we report that there is no $s \in S^x$ such that $s \leftarrow v$.

For the correctness of this algorithm, we show that if there is some $p \in S^x$ with $p \leftarrow v$, then the algorithm returns some solution $s \in S^x$ such that $s \overset{(c\delta,\dots,c\delta)}{\leftarrow} v$. Let

$W = \operatorname{diag}(\omega_1, \ldots, \omega_k)$ and $W' = \operatorname{diag}(\omega'_1, \ldots, \omega'_k)$. Observe that any algorithm for $\mathrm{W}^{\delta}_{\|\cdot\|}$-$\mathcal{O}$ must return a solution if $S^x \neq \emptyset$, which is the case here. So the algorithm must return a solution $s \in S^x$ with $\|W' f^x(s)^T\| \leq \delta \|W' f^x(s')^T\|$ for all $s' \in S^x$. In particular, from Lemma 6.6 we obtain

$$\|W' f^x(s)^T\| \leq \delta \|W' f^x(p)^T\| \leq \hat{c}\delta \|W' v^T\|.$$

Together with $W' = VW$ this yields

$$V\|W f^x(s)^T\| \leq V\hat{c}\delta \|W v^T\|.$$

Now Lemma 6.7 tells us that $f^x(s) \overset{(c\delta,\ldots,c\delta)}{\leftarrow} v$ and so s is returned correctly. For the runtime of the algorithm note that the size of the weights is polynomial in the size of v and the test if $f^x(s) \overset{(c\delta,\ldots,c\delta)}{\leftarrow} v$ is also possible in polynomial time. Furthermore, the particular values for c result from the particular values for $\hat{c}$ and $\tilde{c}$. □

Corollary 6.10. For any k-objective problem $\mathcal{O} = (S, f, \leq)$ and any $\delta \geq 1$ it holds that

$$\mathrm{D}^{\mathrm{k}(\delta,\ldots,\delta)}\text{-}\mathcal{O} \leq^{\mathrm{p}}_{\mathrm{T}} \mathrm{W}^{\delta}\text{-}\mathcal{O}.$$

We want to stress the importance of this result for linear minimization problems like traveling salesperson, minimum spanning tree, minimum perfect matching and the like. Remember that for linear problems, the weighted sum notion is equivalent to the single-objective problem (Proposition 5.3). This result can obviously be extended to the approximate solution notions. Together with the above corollary, this means that for linear single-objective minimization problems that have a PTAS, the k-objective extension is always $k(1+\varepsilon)$-approximable for every $\varepsilon > 0$, i. e. $\mathrm{E}^{\mathrm{k}(1+\varepsilon)}$-$\mathcal{O}$ is polynomial-time solvable. In the next section we will see that, surprisingly, this does not hold for maximization problems (unless P = NP).

Corollary 6.11. For any linear k-objective problem $\mathcal{O} = (S, f, \leq)$, any $\delta \geq 1$ and any $\varepsilon > 0$ it holds that

$$\mathrm{E}^{\mathrm{k}(1+\varepsilon)(\delta,\ldots,\delta)}\text{-}\mathcal{O} \leq^{\mathrm{p}}_{\mathrm{T}} \mathrm{D}^{\mathrm{k}(\delta,\ldots,\delta)}\text{-}\mathcal{O} \leq^{\mathrm{p}}_{\mathrm{T}} \mathrm{W}^{\delta}\text{-}(S, f_1, \leq).$$

Proof. Combine Proposition 5.3, Corollary 6.10 and Theorem 6.3. □

We now show that the reduction from Proposition 6.9 also holds in the other direction (with different factors, though).

Proposition 6.12. For any norm $\|\cdot\|$ on $\mathbb{R}^k$ there is some $c' \geq 1$ such that for any k-objective problem $\mathcal{O} = (S, f, \leq)$ and any $\delta \geq 1$ it holds that

$$\mathrm{W}^{c'\delta}_{\|\cdot\|}\text{-}\mathcal{O} \leq^{\mathrm{p}}_{\mathrm{T}} \mathrm{D}^{(\delta,\ldots,\delta)}\text{-}\mathcal{O}.$$

In particular, if $\|\cdot\|$ is a p-norm then $c' = k^{1/p}$ and $c' = 1$ for the maximum norm.

Proof. Let the instance x and the weights $\omega_1, \dots, \omega_k \in \mathbb{N}$ be inputs for $\mathrm{W}^{\delta}_{\|\cdot\|}$-$\mathcal{O}$. For the sake of clarity, we use $\leftarrow$ instead of $\leq$ in multidimensional contexts. Let $\omega'_i = 1/\omega_i$ if $\omega_i \neq 0$ and otherwise, let ω'_i be some number larger than any possible output of f^x. Using binary search with queries to $\mathrm{D}^{(\delta,\dots,\delta)}$-$\mathcal{O}$ we determine some solution $s \in S^x$ and some $r \in \mathbb{N}$ such that $s \overset{(\delta,\dots,\delta)}{\leftarrow} (r\omega'_1, \dots, r\omega'_k) \in \mathbb{N}^k$ and there is no $s' \in S^x$ such that $s' \leftarrow (r\omega'_1 - 1, \dots, r\omega'_k - 1)$. This means that for any $s' \in S^x$, there is some i such that $f^x_i(s') \geq r\omega'_i$. This can only happen if $\omega_i \neq 0$ and thus we get $\|(\omega_1 f^x_1(s'), \dots, \omega_k f^x_k(s'))^T\|_\infty \geq r$ for all $s' \in S^x$. Regarding s, we get $\omega_i f^x_i(s) \leq \delta r$ for all i, which means that $\|(\omega_1 f^x_1(s), \dots, \omega_k f^x_k(s))^T\|_\infty \leq \delta r \leq \delta \|(\omega_1 f^x_1(s'), \dots, \omega_k f^x_k(s'))^T\|_\infty$ for all $s' \in S^x$ and thus s is a correct output for the problem if the norm is the maximum norm. Otherwise, there are constants $c_1, c_2 > 0$ by the equivalence of norms such that $\|(\omega_1 f^x_1(s), \dots, \omega_k f^x_k(s))^T\| \leq \frac{c_2}{c_1}\delta \|(\omega_1 f^x_1(s'), \dots, \omega_k f^x_k(s'))^T\|$ for all $s' \in S^x$, which completes the first part of the assertion with $c' = \frac{c_2}{c_1}$.

The second part is obtained by observing that $\frac{c_2}{c_1} = k^{1/p}$ for a p-norm. □

Corollary 6.13. For any k-objective problem $\mathcal{O} = (S, f, \leq)$ and any $\delta \geq 1$ it holds that

$$\mathrm{D}^{(\delta,\dots,\delta)}\text{-}\mathcal{O} \equiv^{\mathrm{p}}_{\mathrm{T}} \mathrm{W}^{\delta}_{\|\cdot\|_\infty}\text{-}\mathcal{O}.$$

Note that the factor c' provided by Proposition 6.12 is quite large, especially for the 1-norm. By Theorems 6.4.3 and 6.3.2 we know that (for $\alpha_1 = \dots = \alpha_k$) the much better factor $(1 + \varepsilon)$ is possible by first solving $\mathrm{E}^{\alpha(1+\varepsilon)}$-$\mathcal{O}$ for some $\varepsilon > 1$. This result can be extended to general monotone norms.

Proposition 6.14. Let $\mathcal{O} = (S, f, \leftarrow)$ be a k-objective problem where $\leftarrow = \leq$ or $\leftarrow = \geq$, let $\|\cdot\|$ be some monotone norm on $\mathbb{R}^k$ and $\alpha = (a_1, \dots, a_k)$ with $a_i \geq 1$.

If E^{α}-$\mathcal{O}$ is polynomial-time solvable, then $\mathrm{W}^{\max_i(\alpha_i)}_{\|\cdot\|}$-$\mathcal{O}$ is polynomial-time solvable.

Proof. We show this similarly to Theorem 6.4.3. Suppose all objectives have to be minimized (the proposition can be shown analogously if all objectives have to be maximized). For any instance x and weight vector $\omega = (\omega_1, \omega_2, \dots, \omega_k) \in \mathbb{N}^k$, if $S^x \neq \emptyset$ then there is some $\hat{s} \in S^x$ that minimizes $\|W f^{xT}_i\|$ for $W = \mathrm{diag}(\omega_1, \dots, \omega_k)$. Let S' be a solution of E^{α}-$\mathcal{O}$. Then there must be some $s \in S'$ such that $s \overset{\alpha}{\leftarrow} \hat{s}$, hence

$$f^x_i(s) \leq \alpha_i f^x_i(\hat{s}) \leq \max_i(\alpha_i) f^x_i(\hat{s})$$

for all i, which implies

$$\|W f^x(s)^T\| \leq \|W \max_i(\alpha_i) f^x(\hat{s})^T\| \leq \max_i(\alpha_i) \|W f^x(\hat{s})^T\| \leq \max_i(\alpha_i) \|W f^x(s')^T\|$$

for all $s' \in S^x$. It hence suffices to return a solution $s^* \in S'$ that minimizes $\|W f^{xT}\|$, which can be extracted from S' in polynomial time, because S' has polynomial cardinality. □

Corollary 6.15. The following statements are equivalent for any k-objective problem $\mathcal{O} = (S, f, \leq)$:

- D^{α}-$\mathcal{O}$ is polynomial-time solvable for some $\alpha = (\alpha_1, \dots, \alpha_k)$ with $\alpha_i \geq 1$.
- W^{δ}-$\mathcal{O}$ is polynomial-time solvable for some $\delta \geq 1$.
- $\mathrm{W}^{\delta}_{\|\cdot\|}$-$\mathcal{O}$ is polynomial-time solvable for some norm $\|\cdot\|$ on $\mathbb{R}^k$ and some $\delta \geq 1$.

Further research. By adjusting the individual weights in the reductions, one can change the approximation for D^α-$\mathcal{O}$ in the sense that one can improve the factor for some criteria at the expense of others. It can be further investigated if there are problems where this yields new approximation results, especially for problems where approximation trade-offs are assumed to exist.

6.4 Pareto- versus Scalar Maximization

The previous section showed that for problems where all objectives have to be minimized, approximability results translate from single-objective to multiobjective optimization (Corollary 6.11). We now show the limits of such translations and prove that they are impossible for maximization problems, unless P = NP.

More precisely, we consider 2-MaxClique restricted to instances that consist of an arbitrary graph $G = (V, E, l)$ with labels $(1, 1)$ and two additional nodes x, y that have no connections to other nodes and that have labels $(2n+1, 0)$ and $(0, 2n+1)$, where $n = \#V$.

Definition 6.16. Let $(S', f, \geq) = $ 2-MaxClique and restrict the instances to the set

$$\mathcal{R} = \{(V \cup \{x, y\}, E, l) \mid (V \cup \{x, y\}, E, l) \text{ is an undirected, } \mathbb{N}^2\text{-vertex-labeled graph, } l(x) = (2\#V + 1, 0),\ l(y) = (0, 2\#V + 1), \text{ and } l(v) = (1, 1) \text{ for } v \in V\}.$$

such that we obtain the problem

$$2\text{-MaxClique}_{\text{restr}} = (S, f, \geq) \text{ where } S^{\langle G\rangle} = \begin{cases} S'^{\langle G\rangle} & \text{if } G \in \mathcal{R} \text{ and} \\ \emptyset & \text{otherwise.} \end{cases}$$

Proposition 6.17. For $\mathcal{O} = 2\text{-MaxClique}_{\text{restr}}$ the following holds.

1. W-$\mathcal{O}$ is polynomial-time solvable.
2. There is no $\alpha \in \mathbb{R}^2$ such that E^α-$\mathcal{O}$ is polynomial-time solvable, unless P = NP.
3. There is no $\alpha \in \mathbb{R}^2$ such that D^α-$\mathcal{O}$ is polynomial-time solvable, unless P = NP.

Proof. 1.: On input $(V \cup \{x, y\}, E, l) \in \mathcal{R}$ and $(w_1, w_2) \in \mathbb{N}^k$, the algorithm outputs $\{x\}$ if $w_1 \geq w_2$, and $\{y\}$ otherwise. Note that the special vertices x and y can be easily detected by analyzing l.

2.: Assume that E^α-$\mathcal{O}$ is polynomial-time solvable for some $\alpha \in \mathbb{R}^2$. We may assume $\alpha = (c, c)$ for some $c \geq 1$. We show that Clique is c-approximable which implies P = NP [ALM+92].

Let $G = (V, E)$ be a (nonempty) graph. Define the $2\text{-MaxClique}_{\text{restr}}$ instance $G' = (V \cup \{x, y\}, E, l)$ according to the definition of $\mathcal{R}$. Now consider the solution algorithm for E^α-$\mathcal{O}$ on input G'. Let $m \geq 1$ be the size of the maximal clique in G. Then $S_{\text{opt}}^{\langle G'\rangle}$ contains a solution with value (m, m) and the output of the algorithm must contain some $S \subseteq V \cup \{x, y\}$ such that $c \cdot f^{\langle G'\rangle}(S) \geq (m, m) \geq (1, 1)$. From $f_1^{\langle G'\rangle}(\{y\}) = 0$ and $f_2^{\langle G'\rangle}(\{x\}) = 0$ it follows that $S \neq \{y\}$ and $S \neq \{x\}$. Therefore, $S \subseteq V$, since x and y have no edges with other nodes. Hence $c \cdot \#S \geq m$, i. e., S is a clique of size m/c in G.

3.: Follows from the second part and Theorem 6.3. □

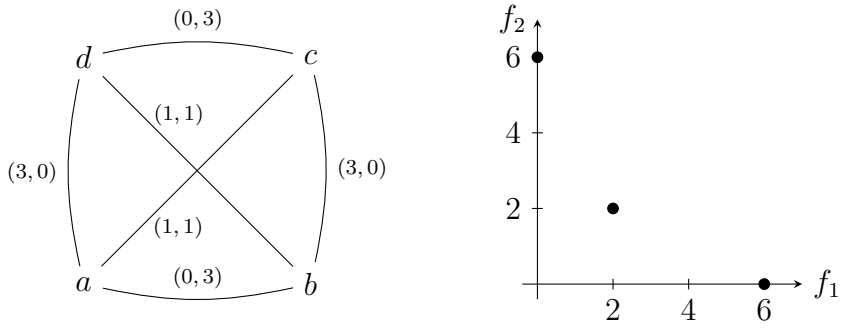


Figure 6.2: The right-hand side shows the values of all solutions to the 2-objective maximum weight perfect matching instance that is shown on the left. There are exactly the three optimal values $(0, 6)$, $(6, 0)$, and $(2, 2)$, but the weighted-sum notion W-$\mathcal{O}$ finds only solutions with values $(0, 6)$ and $(6, 0)$. There is no $\alpha \in \mathbb{R}^2$ such that $(0, 6)$ or $(6, 0)$ weakly α-dominates $(2, 2)$.

The example 2-MaxClique$_{\text{restr}}$ shows the disadvantage of the weighted-sum notion for maximization problems. This effect does not only appear at artificially constructed multiobjective problems. For instance, consider $\mathcal{O}$ to be the 2-objective maximum weight perfect matching problem, which is defined analogously to 2-MM. Here W-$\mathcal{O}$ is polynomial-time solvable. Nevertheless, the instance depicted in Figure 6.2 shows that the solutions for W-$\mathcal{O}$ are not good approximations for the set of Pareto-optimal solutions.

Chapter 7

Approximation Algorithms for Traveling Salesperson Problems

Following the analysis of the structural properties of general multiobjective problems, we now want to focus on a specific problem, the traveling salesperson problem. We take it as an example and want to demonstrate several techniques concerning multiobjective approximation:

Our Results. We show upper bounds for the approximability in section 7.1 by applying approximation algorithms for subproblems and by guessing partial solutions to remove the error introduced by the approximation or move the error from one objective to the other. We thus obtain a deterministic $(2, 2)$-approximation, a randomized $(3/2, 2 + \varepsilon)$- and a randomized $(3/2 + \varepsilon, 2)$-approximation for 2-TSP for every $\varepsilon > 0$.

Using approximation preserving reductions from single-objective problems to multiobjective problems we show the limits of multiobjective approximability in section 7.1.6: An improvement of the approximation of the multiobjective problem would immediately improve the single-objective problem, which seems out of reach or at least requires much more complicated tools that are not yet available for multiobjective optimization. Hereby, the second objective is used to enforce a constraint on the structure of the solution.

The positive and negative results for 2-TSP obtained in section 7.1 are summarized in Table 7.1 on page 100.

Finally, in section 7.2 we show how a result from discrepancy theory can be used as a general tool to transfer ideas from single-objective optimization to multiobjective optimization. We show this for the example of the maximum traveling salesperson problem on directed and undirected graphs, where we obtain a randomized $1/2$-approximation for k-MaxATSP and a randomized $2/3$-approximation for k-MaxSTSP. Note that for maximization problems, we use approximation factors smaller than one. The more unified view on the factors can be obtained by inverting them, i. e. we obtain 2- and $3/2$-approximations, respectively.

For all considered variants of the traveling salesperson problem (minimum with triangle inequality, maximum on undirected and directed graphs), we improve the best known approximation ratios and at the same time give simpler algorithms.

Definitions. We extend the definitions of single-objective approximation to multiobjective approximation in a straightforward way. Note that for multiobjective optimization, the approximation ratio can differ in each objective and thus has to be specified as a vector. Furthermore, it is possible that a single approximation algorithm has several (incomparable) approximation ratios. Some of these definitions have already been given in the previous chapter.

Definition 7.1. Let $\mathcal{O} = (S, f, \leftarrow)$ be a k-objective problem and $\alpha = (\alpha_1, \ldots, \alpha_k) \in \mathbb{R}^k$, such that $\alpha_i \geq 1$ for all $1 \leq i \leq k$.

Some set of solutions $S' \subseteq S^x$ is called an *α-approximation of S^x* if for every $s \in S^x$ there is some $s' \in S'$ such that $s' \overset{\alpha}{\leftarrow} s$. Remember that we have $s' \overset{\alpha}{\leftarrow} s$ if for all $1 \leq i \leq k$, $f_i(s') \leq \alpha_i f_i(s)$ (if $\leftarrow_i = \leq$) or $\alpha_i f_i(s') \geq f_i(s)$ (if $\leftarrow_i = \geq$).

We say that some deterministic or randomized polynomial-time algorithm is an *α-approximation algorithm* for $\mathcal{O}$ if on input x, it computes an α-approximation of S^x and fails with probability at most $1/2$ on each input if it is randomized. Note that it is equivalent to say that such an algorithm solves E^α-$\mathcal{O}$ in (deterministic or randomized) polynomial time.

The vector α above is called an *approximation ratio* of the algorithm or the set S'.

An algorithm is an *FPTAS* (*fully polynomial-time approximation scheme*) for $\mathcal{O}$, if on input x and $\varepsilon > 0$ it computes a $(1+\varepsilon)$-approximation of S^x in time polynomial in $|x| + 1/\varepsilon$. It is called *PTAS* (*polynomial-time approximation scheme*) if its runtime is polynomial in $|x|$.

The analogs of FPTAS and PTAS for randomized algorithms, which are allowed to fail with probability at most $1/2$ on each input, are called *FPRAS* (*fully polynomial-time randomized approximation scheme*) or *PRAS* (*polynomial-time randomized approximation scheme*), respectively.

For maximization problems, it is often more intuitive to consider approximation ratios smaller than 1. If we speak of an approximation ratio α where $0 < \alpha < 1$ (we will especially do this in section 7.2), the above definitions apply to the (componentwise) inverse of α.

7.1 Minimum Traveling Salesperson

7.1.1 Introduction, Related Problems and Known Results

The traveling salesperson problem is one of the oldest combinatorial optimization problems. For a given set of cities, one has to find a shortest round trip that visits each city exactly once. This problem was first mentioned in 1831 as a problem of a traveling salesperson who wants to cover as many locations as possible without visiting locations twice [Voi31]. In the 1950s and 1960s the traveling salesperson problem became increasingly popular in mathematics and computer science.

In the original formulation, the salesperson is not allowed to visit a city more than once. There are two arguments against this restriction: First, it does not make sense for the substantial majority of real-world traveling salesperson problems, including all geometric versions [JP85]. Second, it considerably degrades the approximability of the problem. Therefore, with a minimum loss of generality, one often studies the traveling

salesperson problem where *multiple visits of cities are allowed*. This is equivalent to the *metric traveling salesperson problem* or the *traveling salesperson problem with triangle inequality* (TSP), where one demands that the triangle inequality holds for each triple of cities. A special case of TSP is the *Euclidean* variant, where each city is located at some point in the plane, and the distance function is defined as the Euclidean distance of two cities.

In 1972, a breakthrough was achieved by Karp [Kar72] who proved the NP-hardness of TSP. This shows that the search for a polynomial-time algorithm for TSP is an extremely challenging endeavor and raises the question for good approximation algorithms. For a long time, the best known approximation for TSP and Euclidean TSP was the simple tree-doubling method. In 1976, Christofides [Chr76] improved these results significantly by showing that a combination of a minimum spanning tree with a minimum matching yields a Hamiltonian cycle with approximation ratio $3/2$. After 30 years of research, this basic algorithm is still the best known approximation for TSP.

Let us give a short overview of this algorithm: First it computes a minimum spanning tree of the graph and then a minimum perfect matching of all vertices of odd degree in the tree (this is an even number). Finally, by taking shortcuts if necessary, the minimum spanning tree and the perfect matching are combined to a Hamiltonian cycle. To see that the cycle is a $3/2$-approximation, observe the following: A minimum spanning tree is at most as long as the optimal tour minus some edge. Furthermore, the tour consists of two perfect matchings of all vertices (the edges with odd index and the edges with even index) and thus the length of the minimum perfect matching obtained in the algorithm is at most one half of the length of the tour.

Concerning more restricted variants of the problem, a polynomial-time approximation scheme (PTAS) has been found for the Euclidean TSP by Arora [Aro98] and for $\{0,1\}$-weighted graphs, an approximation ratio better than $3/2$ has been achieved recently by Mömke and Svensson [MS11] and independently by Oveis Gharan, Saberi and Singh [OGSS11].

Regarding lower bounds, Papadimitriou and Vempala [PV06] showed that TSP cannot be approximated with a ratio better than $220/219$, unless P = NP. Another variant of TSP is studied by Papadimitriou and Yannakakis [PY93] who construct a $7/6$-approximation algorithm for TSP(1,2), which is the restriction of TSP where all distances are either 1 or 2.

Two-Objective TSP. The multiobjective traveling salesperson problem was first studied by Gupta and Warburton [GW86]. Angel, Bampis, and Gourvès [ABG04] gave a $3/2$-approximation for the two-objective variant of TSP(1,2). Furthermore, Angel et al. [ABGM05] investigated the non-approximability of this problem. Ehrgott [Ehr00] also studied the multiobjective traveling salesperson problem but the approximation ratio obtained there cannot be compared to the ratios considered here. Manthey and Ram [MR09] gave a $(2+\varepsilon)$-approximation algorithm for multiobjective TSP with componentwise metric cost functions.

Our Results. We extend the usual definition of the multiobjective traveling salesperson problem from Hamiltonian cycles in undirected graphs with componentwise metric cost functions [MR09, Ehr00] to closed walks (that allow vertex repetitions) on edge-labeled

New results:	deterministic	$(2,$	$2)$				Theorem 7.8
	randomized	$(3/2,$	$2+\varepsilon)$				Theorem 7.10
		$(3/2+\varepsilon,$	$2)$				Theorem 7.10
Improvement to		$(3/2-\alpha,$	$\beta)$	yields	$3/2-\alpha$	for TSP	trivial
		$(\frac{1+\sqrt{5}}{2}-\alpha, 2-\varepsilon)$		yields	$\frac{1+\sqrt{5}}{2}-\alpha$	for TSPP_{st}	Theorem 7.12

Table 7.1: Summary of the approximation ratios and connections to single-objective problems obtained for 2-TSP. The variables α, β and ε represent universally quantified positive reals.

hypergraphs. This definition contains the usual definition as a special case and the complication is also worthwhile since, for instance, there are usually no real trade-offs for componentwise-metric cost functions, as we will see later.

Even though we generalize the definition, we can provide more accurate approximations. For 2-TSP we obtain a deterministic 2-approximation, a randomized $(3/2+\varepsilon, 2)$-approximation, and a randomized $(3/2, 2+\varepsilon)$-approximation, where we build on the following known approximation schemes: An FPTAS for multiobjective minimum spanning tree, an FPTAS for multiobjective shortest path, and an FPRAS for multiobjective minimum perfect matching. These were shown by Ravi and Goemans [RG96], Hansen [Han79] and Papadimitriou and Yannakakis [PY00], respectively, while Papadimitriou and Yannakakis gave a generic framework to show all three results. In order to apply these algorithms, we have to extend them to multigraphs. So as a byproduct we provide approximation schemes for the multigraph variants of the mentioned problems.

We present arguments that indicate the hardness of improving our approximation algorithms. Obviously, an improvement to a ratio better than $3/2$ would immediately improve the approximability of the single-objective traveling salesperson problem. This seems very difficult, since there has not been any improvement in the last 35 years. We further give approximation preserving reductions that allow us to translate the well-studied single-objective problem TSPP_{st} to 2-TSP. For TSPP_{st}, the *traveling salesperson path problem*, on input of an edge-labeled undirected complete graph whose weights satisfy the triangle inequality, the task is to find a path connecting two specified vertices and visiting each other vertex exactly once. For a long time, the approximation algorithm by Hoogeveen [Hoo91] for this problem with a ratio of $5/3$ could not be improved. Recently, An, Kleinberg and Shmoys [AKS11] gave an algorithm with approximation ratio $\frac{1+\sqrt{5}}{2} \approx 1.618$. We obtain that certain improvements of the approximation algorithms for 2-TSP force us to still improve this approximation ratio for TSPP_{st}. Table 7.1 on page 100 summarizes the obtained approximation ratios and these arguments.

As a consequence of our results, we obtain a particular interesting situation for 2-TSP (cf. Figure 7.1): We know that 2-TSP is randomized $(3/2, 2+\varepsilon)$-approximable and randomized $(3/2+\varepsilon, 2)$-approximable. It is difficult to improve these approximations with respect to any component, and it is also difficult to obtain a $(\frac{1+\sqrt{5}}{2}-\varepsilon, 2-\varepsilon)$-approximation. However, we have no evidence in favor of or against an (α, β)-approximation where $\frac{1+\sqrt{5}}{2} \leq \alpha, \beta < 2$. The search for such an algorithm remains a challenging open problem.

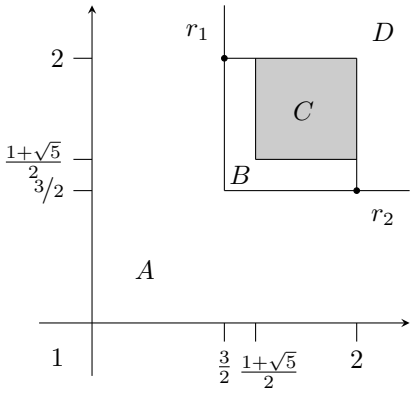


Figure 7.1: Approximation ratios for 2-TSP. An approximation ratio inside A would improve Christofides' approximation and a ratio inside B would improve the approximation by An, Kleinberg and Shmoys [AKS11]. We prove approximation ratios r_1 and r_2, hence area D is of no further interest. However, evidence against approximation algorithms within C is not known.

7.1.2 Definitions

We repeat the definition of k-TSP from section 4.2 and also define the single-objective problem $\mathrm{TSPP}_{\mathrm{st}}$.

Definition 7.2 (k-Objective Traveling Salesperson)**.**

k-TSP $= (S, f, \leq)$ where

- instances are $\mathbb{N}^k$-edge-labeled undirected multigraphs $G = (V, E, l)$,
- $S^{\langle G \rangle} = \{W \subseteq E \mid W \text{ is a closed spanning walk of } G\}$ and
- $f^{\langle G \rangle}(W) = \sum_{e \in W} l(e)$.

Definition 7.3 (Traveling Salesperson Path Problem)**.**

$\mathrm{TSPP}_{\mathrm{st}} = (S, f, \leq)$ where

- instances are $\mathbb{N}$-edge-labeled undirected complete graphs $G = (V, E, l)$ where the labels satisfy the triangle inequality and two distinct vertices $s, t \in V$,
- $S^{\langle G,s,t \rangle} = \{P \subseteq E \mid P \text{ is a closed spanning path in } G \text{ from } s \text{ to } t\}$ and
- $f^{\langle G,s,t \rangle}(P) = \sum_{e \in P} l(e)$.

We now justify why this definition of k-TSP is a suitable generalization of the usual (metric) traveling salesperson problem by first showing that the latter is equivalent to 1-TSP with respect to approximation preserving reductions. For a single objective, parallel edges do not make sense as they can be replaced by the edges with minimal weight. Therefore, 1-TSP is equivalent to the single-objective TSP where multiple visits of cities are allowed. This variant in turn can be reduced to the metric variant by computing the shortest paths between any two vertices in polynomial time. Since the metric traveling salesperson problem is a restriction of 1-TSP, they are equivalent.

The situation changes when we consider TSP with multiple objectives. Here there exist at least three natural variants of different strengths. k-TSP is the most general

variant, which handles arbitrary multigraphs and which allows multiple visits of cities. Metric k-TSP is the restriction of k-TSP where we require that if there is a path between two points in the multigraph, then there is also a direct edge that is at least as short in all components (i. e., a direct connection is always a shortest path between two points). Note that in this variant one can easily avoid multiple visits by taking a shortcut if a node was visited before. Componentwise metric k-TSP is the restriction of metric k-TSP where we require a simple graph that is componentwise metric, i. e., the triangle inequality holds for every component. This variant was studied by Manthey and Ram [MR09], and here again it is easy to avoid multiple visits. Against the background of several variants of different strengths we use the notion k-TSP for the most general variant of the problem. Note that for componentwise metric k-TSP, no real trade-offs are possible: Consider a graph containing two nodes A and B and two cost functions representing road length and toll costs. Since the road length is metric, the single edge between A and B must be the shortest route. At the same time, though, this edge must also be the cheapest connection between A and B for the same reason.

7.1.3 Matching and Spanning Tree Algorithms on Multigraphs

For $k \geq 1$, let multigraph k-MST, multigraph k-MM and multigraph k-SP denote the minimum spanning tree, minimum perfect matching and shortest path problems on $\mathbb{N}^k$-labeled multigraphs defined as the straightforward extensions to multigraphs of the problems defined in section 4.2. Further, define multigraph k-MPM as the problem of finding a minimum cost path matching for the input graph and some given subset of vertices with even cardinality.

It is known that the multiobjective variants of minimum spanning tree (k-MST), shortest path (k-SP), and minimum perfect matching (k-MM) are NP-hard (cf. Proposition 5.4), more specifically, D-$\mathcal{O}$ is NP-hard for all $\mathcal{O} \in \{k\text{-MST}, k\text{-SP}, k\text{-MM}\}$ and $k \geq 2$. Usually, it is only shown that it is NP-hard to decide whether a solution satisfying a specified quality exists. Because of Proposition 3.8 and Theorem 5.21 this is equivalent to actually finding the solution.

Because of this NP-hardness, we need approximation algorithms for the multigraph variants of these problems. We extend known approximation schemes on simple graphs such that they work for multigraphs. In a second step we extend the approximation scheme for multiobjective minimum perfect matching on multigraphs such that it works for multiobjective minimum path matching on multigraphs.

Theorem 7.4 ([RG96, PY00])**.** For any $k \geq 1$ there is an FPTAS for k-MST, an FPRAS for k-MM, and an FPTAS for k-SP.

Corollary 7.5. For any $k \geq 1$ there is an FPTAS for multigraph k-MST, an FPRAS for multigraph k-MM, and an FPTAS for multigraph k-SP.

Proof. We argue for all three problems separately. Remember that the set of vertices incident to a multi-edge e is denoted by $[e]$.

multigraph k-MST: We reduce the problem for $\mathbb{N}^k$-labeled multigraphs to the case of $\mathbb{N}^k$-labeled simple graphs, where the existence of an FPTAS is known by Theorem 7.4.

Let $G = (V, E, c)$ be an $\mathbb{N}^k$-labeled multigraph. We transform G to a simple graph G' by splitting each edge into three parts. More precisely, if an edge $e \in E$ connects the vertices u and v, then we add two new vertices u_e and v_e to the graph and replace e by the three edges $f(e) = \{\{u, u_e\}, \{u_e, v_e\}, \{v_e, v\}\}$. Furthermore, the edge in the middle $\{u_e, v_e\}$ is labeled with $c(e)$, while the remaining two edges are labeled with $(0, \dots, 0)$.

Formally, $G' = (V \cup U, E', c')$ where $U = \{v_e \mid e \in E, v \in [e]\}$, $E' = \bigcup_{e \in E} f(e)$, and $c' \colon E' \to \mathbb{N}^k$ such that

$$c'(e') = \begin{cases} c(e) & \text{if } [e'] = \{u_e, v_e\} \text{ for some } u, v \in V, e \in E \text{ and} \\ (0, \dots, 0) & \text{if } [e'] \not\subseteq U. \end{cases}$$

We extend f to subsets $E_1 \subseteq E$ by $f(E_1) = \bigcup_{e \in E_1} f(e)$. So f translates subsets $E_1 \subseteq E$ into subsets $E_1' \subseteq E'$. For the converse translation, let

$$g(E_1') = \{e \in E \mid \{u_e, v_e\} \in E_1' \text{ for some } u, v \in V\}$$

for $E_1' \subseteq E'$. Observe that f and g respect the sum of the labels, i. e.,

$$c(E_1) = c'(f(E_1)) \quad \text{and} \quad c'(E_1') = c(g(E_1')). \tag{7.1}$$

Each path in some $E_1' \subseteq E'$ that starts and ends in nodes from V induces a path in $g(E_1')$ with the same start and end nodes. Therefore,

$$E_1' \text{ is connected and covers } V \quad \Rightarrow \quad g(E_1') \text{ is connected and covers } V. \tag{7.2}$$

We describe the FPTAS for multigraph k-MST on input $G = (V, E, c)$ and $\varepsilon > 0$: Transform G into G' as described above and run the FPTAS for k-MST [PY00] on input G' and ε. We obtain a $(1 + \varepsilon)$-approximation $\mathcal{A}$ of all minimal spanning trees of G'. For each $T' \in \mathcal{A}$, compute $g(T')$, prune this graph as long as it contains any cycles, and output the resulting tree.

The running time of the algorithm is polynomial in $|G|$ and $1/\varepsilon$.

Let T be a minimal spanning tree of $G = (V, E, c)$. We argue that the algorithm above outputs a $(1+\varepsilon)$-approximation of T. Observe that $f(T) \cup \{e' \in E' \mid [e'] \not\subseteq U\}$ is a spanning tree of G' with costs $c'(f(T)) = c(T)$. So $\mathcal{A}$ contains a spanning tree T' of G' such that $c'(T') \leq (1 + \varepsilon) \cdot c(T)$. By (7.2), $g(T')$ is connected and by (7.1), $c(g(T')) = c'(T') \leq (1 + \varepsilon) \cdot c(T)$. Hence the algorithm outputs a spanning tree of G with costs at most $(1 + \varepsilon) \cdot c(T)$.

multigraph k-MM: We again reduce this problem to the case of minimal perfect matchings in an $\mathbb{N}^k$-labeled simple graph, where the existence of an FPTAS is known by Theorem 7.4. For this, let $G = (V, E, c)$ be an $\mathbb{N}^k$-labeled multigraph. We transform G to exactly the same $\mathbb{N}^k$-labeled simple graph $G' = (V \cup U, E', c'')$ as in the proof for multigraph k-MST, only the cost function c'' is defined differently. So recall U and E' from the first part. The label of the original edge $c(e)$ is put on both outer edges, while the edge in the middle is labeled with zero. More formally:

$$c''(e') = \begin{cases} c(e) & \text{if } [e'] = \{v, v_e\} \text{ for some } v \in V, e \in E \text{ and} \\ (0, \dots, 0) & \text{if } [e'] \subseteq U \end{cases}$$

We also define a different converse translation h (which is not inverse to f):

$$h(E_1') = \{e \in E \mid \{u_e, v_e\} \notin E_1' \text{ for } u, v \in V \text{ with } \{u, v\} = [e]\}$$

for $E_1' \subseteq E'$.

The FPRAS for multigraph k-MM on input $G = (V, E, c)$ and $\varepsilon > 0$ works as follows: Transform G into G' as described above and run the FPRAS for k-MM [PY00] on input G' and ε. We obtain a $(1+\varepsilon)$-approximation $\mathcal{A}$ of all minimal perfect matchings G'. For each $M' \in \mathcal{A}$, output $h(M')$.

The running time of the algorithm is polynomial in $|G|$ and $1/\varepsilon$. It remains to show that the returned edge sets are in fact perfect matchings and that they approximate the minimal perfect matchings of G with probability at least $1/2$.

For the first part, consider some perfect matching $M' \subseteq E'$ of G'. Observe that for any $e \in E$ with $[e] = \{u, v\}$, we have $\{u, u_e\}, \{v_e, v\} \in M' \iff \{u_e, v_e\} \notin M'$. So since any vertex $v \in V$ must be matched exactly once in M', there is exactly one $e \in E$ such that $\{v, v_e\} \in M'$. Since u is uniquely determined by $\{u, v\} = [e]$, we get that there is exactly one $u \in V$ such that $\{u, v\} \in h(M')$ and thus $h(M')$ is a perfect matching of G'.

For the second part, first note that for any perfect matching M' of G' we have

$$c''(M') = 2c(h(M')). \tag{7.3}$$

Now let M be a perfect matching of G and consider $M' = \{\{v, v_e\} \mid v \in [e], e \in M\} \cup \{\{u_e, v_e\} \mid \{u, v\} = [e], e \in E - M\}$. Obviously, M' is a perfect matching of G' and $h(M') = M$. So we have $c''(M') = 2c(M)$, and the output of the FPRAS must contain some perfect matching $\tilde{M}'$ of G' such that $c''(\tilde{M}') \leq (1+\varepsilon)c''(M') = 2(1+\varepsilon)c(M)$ with probability at least $1/2$. From $\tilde{M}'$, we obtain a perfect matching $\tilde{M} = h(\tilde{M}')$ of G such that $c(\tilde{M}) = \frac{1}{2}c''(\tilde{M}') \leq (1+\varepsilon)c(M)$ and thus the assertion is proved.

multigraph k-SP: For multigraph shortest path, we use exactly the same construction as in the proof for multigraph k-MM. So recall the notions from the part about multigraph k-MST. The algorithm works as follows:

On input of a multigraph (V, E, c), $s, t \in V$ and $\varepsilon > 0$, construct $G' = (V \cup U, E', c')$ as in the first part, run the FPTAS for k-SP [PY00] on G', s, t and ε, apply g to the paths found by the FPTAS and return the results.

The algorithm obviously runs in polynomial time and solves the problem because of the properties of f, g and c already noted in the first part. □

The FPRAS for multigraph k-MPM that is stated in the following theorem is based on the approximation schemes for multigraph k-SP and multigraph k-MM.

Theorem 7.6. For any $k \geq 1$ there is an FPRAS for multigraph k-MPM.

Proof. Let $G = (V, E, c)$ be an $\mathbb{N}^k$-labeled multigraph, $U \subseteq V$ be a set of even cardinality and $\varepsilon > 0$.

We start by constructing an $\mathbb{N}^k$-labeled multigraph $G' = (U, E', c')$ by approximately computing shortest paths between vertices of U and letting each path be an edge in G' between its endpoints. More formally: For every two-element subset $\{s, t\}$ of U, run the FPTAS for multigraph k-SP (cf. Corollary 7.5) on G, s, t and ε to obtain the approximate Pareto set $\{p_1^{\{s,t\}}, \dots, p_{m_{\{s,t\}}}^{\{s,t\}}\}$ of shortest paths between s and t in G. Add each of these paths as a single edge to the graph to obtain the set of edges $E' = \{(\{s, t\}, i) \mid s, t \in U, s \neq t \text{ and } 1 \leq i \leq m_{\{s,t\}}\}$ and set $c'(\{s, t\}, i) = c(p_i^{\{s,t\}})$ for $(\{s, t\}, i) \in E'$.

Now run the FPRAS for multigraph k-MM (cf. Corollary 7.5) on G' and ε and obtain an approximate Pareto set of matchings $\mathcal{M}$. Finally, for each perfect matching $M \in \mathcal{M}$, return $\{p_i^{\{s,t\}} \mid (\{s, t\}, i) \in M\}$.

The running time of the algorithm is polynomial in $|G|$ and $1/\varepsilon$, and the returned sets are path matchings since every vertex $s \in U$ is matched by exactly one edge in M and thus by exactly one path. Concerning the approximation ratio, let P be a path matching of U in G. Every path $p \in P$ with endpoints $s, t \in U$ is approximated by the FPTAS for multigraph k-SP, which means that there is some $1 \leq i \leq m_{\{s,t\}}$ such that $c(p_i^{\{s,t\}}) \leq (1 + \varepsilon)c(p)$. For $\tilde{P}$ being the set of these (approximately) shortest paths $p_i^{\{s,t\}}$ for all paths $p \in P$, we obtain $c(\tilde{P}) \leq (1 + \varepsilon)c(P)$. Furthermore, $\tilde{P}$ is a path matching and thus corresponds to a perfect matching M of G' with the same costs. This perfect matching is approximated by a perfect matching $\tilde{M}$ using the FPRAS for multigraph k-MM. For the path matching $\tilde{P}'$ finally obtained from $\tilde{M}$, we have the inequality

$$\begin{aligned} c(\tilde{P}') &= c'(\tilde{M}) \\ &\leq (1 + \varepsilon)c'(M) = (1 + \varepsilon)c(\tilde{P}) \\ &\leq (1 + \varepsilon)(1 + \varepsilon)c(P) = (1 + 2\varepsilon + \varepsilon^2)c(P) \\ &\leq (1 + 3\varepsilon)c(P) \end{aligned}$$

which means that the algorithm described above is an FPRAS for k-MPM. □

In the algorithms that will follow, we will explicitly call the approximation schemes for multigraph 2-MST and multigraph 2-MPM and denote them by `2-MST-Approx`(V, E, c, ε) and `2-MPM-ApproxRand`$(V, E, c, U, \varepsilon)$, where (V, E, c) is an $\mathbb{N}^k$-labeled multigraph, $U \subseteq V$ are the vertices to match (even cardinality), and ε is the approximation factor. On a single input, the randomized approximation `2-MPM-ApproxRand` will be called multiple times. Thus we assume that this approximation is amplified in a way such that the probability that *all* calls succeed is at least $1/2$.

7.1.4 Deterministic Approximation

Our deterministic 2-approximation for 2-TSP is inspired by Christofides' approximation for TSP. In contrast to the single-objective problem, we cannot assume that the instances of 2-TSP are metric and thus have to replace the perfect matching for the odd degree vertices in the tree by a path matching. Furthermore, we do not compute the minimal path matching but only show that a suitable path matching can be extracted *deterministically* from an approximate two-objective minimum spanning tree. More precisely, we transform a spanning tree into a path matching of at most the same costs. Thus we avoid the randomness of the approximation algorithm for two-objective minimum path matching

(2-MPM). Although this extracted path matching is by far not optimal, it suffices to improve the approximation, since at present, the bottleneck of approximations for 2-TSP is not the method of finding a good path matching, but rather the argument that it exists. Lemma 7.7 shows that Algorithm 7.1 computes a suitable path matching in polynomial time which is in turn used by Algorithm 7.2 to compute the $(2, 2)$-approximation for 2-TSP.

Algorithm 7.1: `match`$(T, U[, n])$

`Input` : tree T, set of nodes U, node n (default: root of T)
`Output`: path matching on U in T (if possible)

```
1 X := ({n} ∩ U) ∪ ⋃_{c child of n} match(T, c, U);
2 while #X ≥ 2 do
3     remove two nodes s, t from X and output the unique path from s to t
4 return X
```

Lemma 7.7. Let $G = (V, E, c)$ be some $\mathbb{N}^k$-labeled multigraph for $k \geq 1$, $T \subseteq E$ a spanning tree of G and $r \in V$. Then, for any $U \subseteq V$ of even cardinality, `match`(T, U) finds in polynomial time a path matching M of U in T such that $c(M) \leq c(T)$.

Proof. We show that each edge of the path matching is an edge of the tree and is used exactly once, which suffices to show $c(M) \leq c(T)$.

We can assume an arbitrary node r to be the root of T and then define a suitable parent and child relation. Note that outputting a path (i. e. inserting it into the final set of paths returned by the algorithm) is different from returning a set of nodes to the caller. Because of the condition on the loop, it always holds $\#X \leq 1$ when X is returned. Since each recursive call handles its own subtree, it can be shown by induction that when $\{v\}$ is returned by a recursive call, no edge on the path from v to the root r has been output yet. This means that when the path between two nodes s, t in the recursive call `match`(T, n, U) is output, none of its edges has been output yet (n is the common ancestor of s and t) and none of them will be output again. This shows that $c(M) \leq c(T)$.

Observe that all nodes are matched if $\#U$ is even. Furthermore, the algorithm runs in polynomial time since we have at most one recursive call for each node in the tree. □

Algorithm 7.2: `2-TSP-ApproxDet`(V, E, c)

`Input` : $\mathbb{N}^2$-labeled multigraph (V, E, c)
`Output`: set of closed spanning walks of (V, E, c)

```
1 ε := 1/(2#V);
2 T := 2-MST-Approx(V, E, c, ε);
3 foreach (T_1, T_2) ∈ T × T do
4     U := {v ∈ V | deg_{T_1}(v) is odd};
5     M := match(T_2, U);
6     output closed spanning walk of (V, E) using T_1 and M;
```

Theorem 7.8. 2-TSP is $(2, 2)$-approximable.

Proof. Let $G = (V, E, c)$ be a $\mathbb{N}^2$-labeled multigraph and W an arbitrary closed spanning walk of (V, E). We show that `2-TSP-ApproxDet`(V, E, c) outputs a closed spanning walk W_{approx} such that $c(W_{\text{approx}}) \leq 2c(W)$.

As in Algorithm 7.2, let $m := \#V$ and $\varepsilon := \frac{1}{2m}$. We split W into contiguous subwalks $W_1, \ldots, W_m$ such that every vertex is at one end of at least one of the subwalks. This can be achieved by letting every subwalk start at the first occurrence of some vertex in W. For every $i \in \{1, 2\}$ there is some $1 \leq p_i \leq m$ such that $c_i(W_{p_i}) \geq \frac{1}{m} c_i(W)$. By removing W_{p_i} from W, the multiset of edges E_i thus obtained is connected and covers every vertex of V. Thus (V, E) has spanning trees T_i' with no higher costs than E_i, which means that $c(T_1') \leq ((1 - 1/m)c_1(W), c_2(W))$ and $c(T_2') \leq (c_1(W), (1 - 1/m)c_2(W))$. The FPTAS for the minimum spanning tree, `2-MST-Approx`(V, c, ε), provides an ε-approximation of every spanning tree of G. So T_1' and T_2' are approximated by say T_1 and T_2 such that $c(T_1) \leq (1 + 1/2m)c(T_1')$ and $c(T_2) \leq (1 + 1/2m)c(T_2')$. Consider the respective loop iteration for the rest of the proof.

The number of vertices of odd degree in an undirected graph is even, so $c(M) \leq c(T_2)$ by Lemma 7.7. The closed spanning walk W_{approx} can be easily constructed since all vertices of odd degree in T_1 are matched by edges in M and thus every vertex has even degree when the edges of T_1 and M are used. Concerning the costs we obtain

$$\begin{aligned} c\left(W_{\text{approx}}\right) &\leq c\left(T_1\right) + c\left(M\right) \leq c\left(T_1\right) + c\left(T_2\right) \leq \left(1 + \frac{1}{2m}\right)\left(c\left(T_1'\right) + c\left(T_2'\right)\right) \\ &\leq \left(1 + \frac{1}{2m}\right)\left(1 - \frac{1}{m} + 1\right) c\left(W\right) = \left(2 - \frac{1}{2m^2}\right) c\left(W\right) < 2c\left(W\right). \end{aligned}$$

It remains to show that Algorithm 7.2 runs in polynomial time. The runtime of the FPTAS `2-MST-Approx` is polynomially bounded in $m + \frac{1}{\varepsilon} = 3m$. Thus, the cardinality of $\mathcal{T}$ itself is bounded by a polynomial in m, say p. For each of the p^2 combinations of spanning trees, the steps 4–6 can be carried out in polynomial time (cf. Lemma 7.7). Hence Algorithm 7.2 is a polynomial-time algorithm. □

7.1.5 Randomized Approximation

The randomized algorithm `2-TSP-ApproxRand`$_\varepsilon$ that is given below provides both a $(3/2 + \varepsilon, 2)$-approximation and a $(3/2, 2 + \varepsilon)$-approximation for 2-TSP. This algorithm is an enhanced variant of a randomized approximation for the componentwise metric 2-TSP that was studied by Manthey and Ram [MR09]. First, it computes approximations of the minimum spanning trees, then considers the vertices that have odd degree in a single tree, computes approximations of the minimum path matchings of these vertices, and finally pairwise combines all trees with all suitable matchings which results in a set of closed spanning walks. A precise analysis provides approximation ratios that are better than the ones stated by Manthey and Ram [MR09], even though the new algorithm manages a more general variant of the problem.

The use of the FPTAS for the two-objective minimum spanning tree problem is essential, as it allows us to reduce the error far enough such that it is dominated by the costs of a contiguous subwalk of an optimal walk. This makes it possible to remove an ε-error in one of the two objectives.

Lemma 7.9. For every $\varepsilon > 0$, Algorithm 7.3 runs in polynomial time.

Algorithm 7.3: 2-TSP-ApproxRand$_\varepsilon(V, E, c)$

```
Input : ℕ²-labeled multigraph (V, E, c)
Output: set of closed spanning walks of (V, E, c)
1 m := #V; ε₁ := ε/m²; ε₂ := ε/(2m);
2 P := 2-MST-Approx(V, E, c, ε₁);
3 foreach T ∈ P do
4     U := {v ∈ V | deg_T(v) is odd};
5     A := 2-MPM-ApproxRand(V, E, c, U, ε₂);
6     foreach M ∈ A do
7         output closed spanning walk using the edges of T and M;
```

Proof. Since 2-MST-Approx is an FPTAS, its running time is polynomial in $n+\frac{1}{\varepsilon_1} = n+\frac{m^2}{\varepsilon}$ where n is the size of the input (V, E, c). So we can obtain $\mathcal{P}$ in polynomial time and $\mathcal{P}$ contains only polynomially many elements. This means that the first loop is iterated polynomially often. 2-MPM-ApproxRand runs in polynomial time in $\frac{1}{\varepsilon_2}$ plus the length of (V, E, c, U) and thus also in n. This in turn means that $\mathcal{A}$ contains only polynomially many matchings, so the second loop is iterated only polynomially often. The operation in line 7 can obviously be carried out in polynomial time and thus the whole algorithm runs in polynomial time. □

Theorem 7.10. For every $\varepsilon > 0$, 2-TSP is randomized $(3/2 + \varepsilon, 2)$-approximable and randomized $(3/2, 2 + \varepsilon)$-approximable.

Proof. We first show that Algorithm 7.3 computes a $(3/2 + \varepsilon, 2)$-approximation for 2-TSP. By Lemma 7.9 the algorithm runs in polynomial time, so it remains to show the approximation ratio and the success probability.

Let $\varepsilon > 0$ and assume $\varepsilon \leq 1$ (otherwise, just call the algorithm with $\varepsilon = 1$). Let (V, E, c) be some $\mathbb{N}^2$-labeled multigraph and W be an arbitrary closed walk of (V, E). We can assume that $\#V \geq 2$. We show that Algorithm 7.3 computes a $(3/2 + \varepsilon, 2)$-approximation for W with probability at least $1/2$. Define m, ε_1 and ε_2 as in the algorithm. As in the proof of Theorem 7.8, we can again argue that (V, E, c) has a spanning tree T' with $c(T') \leq (c_1(W), \frac{m-1}{m} c_2(W))$. Hence, the algorithm 2-MST-Approx finds a spanning tree T with costs

$$c(T) \leq \left(1 + \frac{\varepsilon}{m^2}\right)\left(c_1(W), \left(1 - \frac{1}{m}\right) c_2(W)\right).$$

Consider the iteration that handles T and let $U \subseteq V$ be the vertices of odd degree in T (U has even cardinality). From W we can easily find two path matchings M_1 and M_2 of U in (V, E, c) such that $c(M_1) + c(M_2) \leq c(W)$: For each $u \in U$, fix the first occurrence in W, and cut W at each of those $\#U$ positions to obtain $\#U$ subwalks $S_1, \ldots, S_{\#U}$. For each i, we remove any cycles from S_i and obtain a path P_i with $c(P_i) \leq c(S_i)$. Then, both $M_1 = \{P_i \mid i \text{ even}\}$ and $M_2 = \{P_j \mid j \text{ odd}\}$ are path matchings of U in (V, E, c). Hence, there is some path matching M' of U in (V, E, c) such that $c(M') \leq (\frac{1}{2} c_1(W), c_2(W))$.

There must be some approximate minimum path matching M in $\mathcal{A}$ (with probability at least $1/2$) such that $c(M) \leq (1 + \frac{\varepsilon}{2m})(\frac{1}{2} c_1(W), c_2(W))$. The iteration that considers M now outputs a spanning walk W_{approx} with the following costs (note that $m \geq 2$ and

$\varepsilon \leq 1$):

$$\begin{aligned} c_1(W_{\text{approx}}) &\leq c_1(T) + c_1(M) \\ &\leq \left(1 + \frac{\varepsilon}{m^2}\right) c_1(W) + \left(1 + \frac{\varepsilon}{2m}\right) \frac{1}{2} c_1(W) \leq \left(\frac{3}{2} + \varepsilon\right) c_1(W) \end{aligned}$$

$$\begin{aligned} c_2(W_{\text{approx}}) &\leq c_2(T) + c_2(M) \\ &\leq \left(\left(1 + \frac{\varepsilon}{m^2}\right)\left(1 - \frac{1}{m}\right) + \left(1 + \frac{\varepsilon}{2m}\right)\right) c_2(W) \\ &\leq \left(\left(1 + \frac{1}{m^2}\right)\left(1 - \frac{1}{m}\right) + \left(1 + \frac{1}{2m}\right)\right) c_2(W) \\ &\leq \left(1 - \frac{1}{m} + \frac{1}{m^2} + 1 + \frac{1}{2m}\right) c_2(W) \\ &\leq \left(2 - \frac{1}{2m} + \frac{1}{m^2}\right) c_2(W) \\ &\leq 2\, c_2(W) \end{aligned}$$

Using symmetric arguments, the algorithm also finds a spanning tree T and a path matching M of the odd-degree vertices of T such that $c(T) \leq (1 + \frac{\varepsilon}{m^2})(\frac{m-1}{m} c_1(W), c_2(W))$ and $c(M) \leq (1 + \frac{\varepsilon}{2m})(\frac{1}{2} c_1(W), c_2(W))$. By combining these, we obtain a spanning walk W_{approx} such that

$$\begin{aligned} c_1(W_{\text{approx}}) &\leq c_1(T) + c_1(M) \\ &\leq \left(\left(1 + \frac{\varepsilon}{m^2}\right)\left(1 - \frac{1}{m}\right) + \left(1 + \frac{\varepsilon}{2m}\right)\frac{1}{2}\right) c_1(W) \\ &\leq \left(\left(1 + \frac{1}{m^2}\right)\left(1 - \frac{1}{m}\right) + \left(1 + \frac{1}{2m}\right)\frac{1}{2}\right) c_1(W) \\ &\leq \left(1 - \frac{1}{m} + \frac{1}{m^2} + \frac{1}{2} + \frac{1}{4m}\right) c_1(W) \\ &\leq \left(\frac{3}{2} - \frac{3}{4m} + \frac{1}{m^2}\right) c_1(W) \\ &\leq \frac{3}{2}\, c_1(W) \end{aligned}$$

and

$$\begin{aligned} c_2(W_{\text{approx}}) &\leq c_2(T) + c_2(M) \\ &\leq \left(\left(1 + \frac{\varepsilon}{m^2}\right) + \left(1 + \frac{\varepsilon}{2m}\right)\right) c_2(W) \\ &\leq \left(\left(1 + \frac{\varepsilon}{4}\right) + \left(1 + \frac{\varepsilon}{4}\right)\right) c_2(W) \\ &\leq (2 + \varepsilon)\, c_2(W)\,. \end{aligned}$$

Note that the probability that all Pareto-optimal points are appropriately approximated can be smaller than $\frac{1}{2}$. This can be compensated easily by appropriately amplifying the success probability of `2-MPM-ApproxRand`. □

7.1.6 Lower Bound Arguments

As explained in section 7.1.1 (cf. also Figure 7.1 on page 101), we discuss implications of improvements of the approximation ratio of 2-TSP obtained in Theorem 7.10. We show that

- an improvement of the second component, i. e. to $(\frac{1+\sqrt{5}}{2} - \varepsilon, 2 - \varepsilon)$ would improve the currently best known approximation for TSPP_{st} by An, Kleinberg and Shmoys [AKS11].
- an improvement of the first component, i. e. to $(3/2 - \varepsilon, 2 + \varepsilon)$ would improve the currently best known approximation for TSP by Christofides [Chr76].

On the other hand, we have no evidence in favor of or against an (α, β)-approximation where $\frac{1+\sqrt{5}}{2} \leq \alpha, \beta < 2$. Note that this approximation ratio is incomparable to our results.

The second result is obtained by an easy observation, while the first result follows from an approximation preserving reduction from a single-objective to a two-objective problem where the second objective is used to restrict the set of possible solution.

Note that since we show this lower bound argument for componentwise metric 2-TSP it is equally true for the less restrictive variants of TSP.

Proposition 7.11. Let $\alpha > 1$ and $\varepsilon > 0$. The following holds for deterministic and randomized approximations: If componentwise metric 2-TSP is $(3/2 - \varepsilon, \alpha)$-approximable, then TSP is $(3/2 - \varepsilon)$-approximable.

Proof. This is immediate by adding a dummy objective. □

Theorem 7.12. Let $\alpha > 1$ and $\varepsilon > 0$. The following holds for deterministic and randomized approximations: If componentwise metric 2-TSP is $(\alpha, 2 - \varepsilon)$-approximable, then TSPP_{st} is α-approximable.

Proof. Let $\mathcal{A}$ be an algorithm that on input of a complete $\mathbb{N}^2$-labeled simple graph (V, E, c) with componentwise metric distance function c returns an $(\alpha, 2 - \varepsilon)$-approximation for componentwise metric 2-TSP for some $\alpha \geq 1$ and some $\varepsilon > 0$. Let $((V, E, c'), s, t)$ be an arbitrary TSPP_{st}-instance where $V = \{s, t, v_1, \ldots, v_k\}$. Since TSPP_{st} is a single-objective problem, we can assume that the graph is simple and complete and c' is metric. We will construct an instance I of componentwise metric 2-TSP for $\mathcal{A}$ that depends on some natural number $r > 1/\varepsilon$ (cf. Figure 7.2). We start by creating a copy $V' = \{s', t', v'_1, \ldots, v'_k\}$ of V and denote by $v' \in V'$ the copy of $v \in V$. Furthermore, we create "bridges" from s to s' and from t to t' using $r - 1$ additional vertices each, which will be called $B^s = \{s = b^s_0, b^s_1, \ldots, b^s_{r-1}, s' = b^s_r\}$ and $B^t = \{t = b^t_0, b^t_1, \ldots, b^t_{r-1}, t = b^t_r\}$. So the vertices of our componentwise metric 2-TSP instance are $V \cup V' \cup B^s \cup B^t$ and we have all possible edges since the graph must be complete. The labeling function will be defined as follows. First, we define it directly for some of the edges:

- for $e \subseteq V$ or $e \subseteq V'$, we set $c(e) := (c'(e), 0)$
- for $e = \{b^s_i, b^s_{i+1}\}$ or $e = \{b^t_i, b^t_{i+1}\}$, we set $c(e) := (0, 1)$

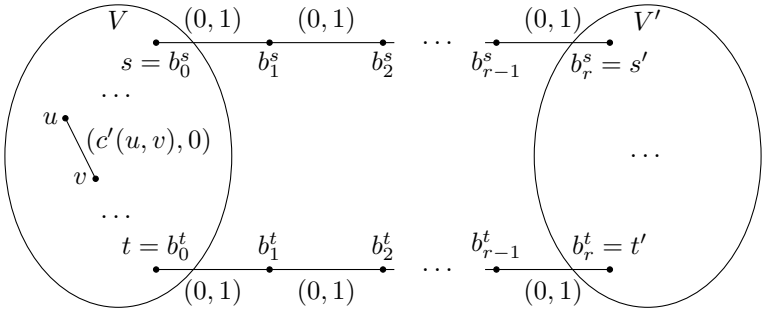


Figure 7.2: Creating an instance of componentwise metric 2-TSP from an instance $((V, E, c'), s, t)$ of TSPP_{st}. We first make a copy V' of V and, for each $u, v \in V$, we set $c(u, v) = (c'(u, v), 0)$ and $c(u', v') = (c'(u, v), 0)$. We further connect s with s' and t with t' by $r - 1$ bridge vertices b_i^s, b_i^t for $1 \leq i \leq r - 1$, and distribute the distance of $c(s, s') = c(t, t') = (0, r)$ equally among the bridge edges.

For all other vertex pairs and for each component, we indirectly define the distance as the length of the shortest path between these vertices using only edges from the above two categories.

In order to show that the functions c_1 and c_2 are metric, we have to show that the directly defined distance between any two vertices is not longer than any path between them that uses edges with directly defined distances. For c_2, this is obviously the case.

We now argue for c_1. Let $u, v \in V$ and consider a path between u and v. If the path does not use the bridges and V', then it cannot be shorter than $c'(u, v) = c_1(u, v)$, since c' is metric on V. So let us assume that the path uses the bridges and V'; w.l.o.g. the s-bridge is used first. Observe that the length of the path is at least $c'(u, v) = c_1(u, v)$. The case where $u, v \in V'$ is of course symmetric and this property obviously holds for bridge edges, since they have distance 0. Hence c_1 is metric.

Let P be the c'-shortest Hamiltonian path between s and t in V and P' its (reversed) copy in V'. $P \cup \{\{t, b_1^t\}, \ldots, \{b_{r-1}^t, t'\}\} \cup P' \cup \{\{s', b_{r-1}^s\}, \ldots, \{b_1^s, s\}\}$ is obviously a Hamiltonian cycle in the new graph with costs $(2c'(P), 2r)$. Since it is a valid solution, $\mathcal{A}$ must return an $(\alpha, 2 - \varepsilon)$-approximation of it. So $\mathcal{A}$ must return a solution S such that $c_2(S) \leq 4r - 2\varepsilon r$. We will now show that from S we can extract a Hamiltonian s-t-Path (in V) with length of at most $\alpha \cdot c'(P)$.

Let $E_{B^t} := \{\{b_{i-1}^t, b_i^t\} \mid 1 \leq i \leq r\} \cup \{\{b_i^t, b_{i-1}^t\} \mid 1 \leq i \leq r\}$ be the "simple" edges of the t-bridge and E_{B^s} be the analogously defined "simple" edges of the s-bridge. We can modify S such that edges crossing the set boundaries of V, V', B^t and B^s are replaced by a detour via the corresponding "portal" s, t, s', or t', possibly using a bridge. In other words, we only allow edges from the set $\{\{u, v\} \mid \text{either } u, v \in V \text{ or } u, v \in V'\} \cup E_{B^s} \cup E_{B^t}$. This modification does not raise any costs, as the costs for edges crossing these boundaries are in fact defined by taking detours via the portals. Hence, from now on we may assume that S only uses edges from $\{\{u, v\} \mid \text{either } u, v \in V \text{ or } u, v \in V'\} \cup E_{B^s} \cup E_{B^t}$.

We will now argue that S uses each bridge exactly once. We denote by $u(x, y)$ the number of times the edge $\{x, y\}$ is used in S and by $d(v)$ the degree of a vertex v in S considered as a multigraph. Furthermore, $d(V) = u(s, b_1^s) + u(t, b_1^t)$ and $d(V') = u(b_{r-1}^s, s') + u(b_{r-1}^t, t')$ are the "degrees" of the subgraphs V and V'.

Observe the following facts.

1. The degrees $d(v)$ for every vertex v and $d(V)$ and $d(V')$ are all even.

2. The parity of $u(e)$ is the same for all edges $e \in E_{B^s} \cup E_{B^t}$.

3. There can be at most one edge $e \in E_{B^s} \cup E_{B^t}$ such that $u(e) = 0$.

4. All bridge edges $e \in E_{B^s} \cup E_{B^t}$ have odd usage count $u(e)$.

If $u(e) > 1$ for some bridge edge e, we can always remove two uses of this edge from S without destroying the property of S being a closed spanning walk. So we may assume S to be of the form such that every bridge edge is used exactly once. This means that S starts at s, visits every vertex in V, goes to t, uses the bridge to t', visits every vertex in V', goes to s' and uses the bridge back to s. So S restricted to V is a Hamiltonian path from s to t and another one can be obtained by restricting S to V'. We can thus extract a Hamiltonian path with length at most $\frac{1}{2}c_1(S) \leq \frac{1}{2} \cdot 2 \cdot c'(P) \cdot \alpha = c'(P) \cdot \alpha$. This is an α-approximation for the TSPP_{st}-instance $((V, E, c'), s, t)$. □

Corollary 7.13. Let $\varepsilon > 0$. The following holds for deterministic and randomized approximations: If componentwise metric 2-TSP is $(\frac{1+\sqrt{5}}{2} - \varepsilon, 2 - \varepsilon)$-approximable, then TSPP_{st} is $(\frac{1+\sqrt{5}}{2} - \varepsilon)$-approximable.

7.2 Maximum TSP and the Discrepancy Technique

7.2.1 Introduction, Related Problems and Known Results

A common strategy for (single-objective) approximation algorithms is a variation of the pigeonhole principle: One first shows that there is a fixed number (say r) of alternatives whose weights sum to x. Then, the heaviest of these alternatives has at least the weight $1/r \cdot x$. This strategy is for example used by a straightforward $1/2$-approximation algorithm for the (single-objective) maximum weighted satisfiability problem 1-SAT (cf. Definition 4.10). There, the input is a set of clauses with weights and the goal is to find an assignment of the variables that maximizes the sum of the weights of the satisfied clauses. The two alternatives are an arbitrary assignment and its complementary assignment. Note that the sets of satisfied clauses need not be disjoint for the strategy to work.

Another application of this idea appears in the algorithm of Christofides, which provides a $3/2$-approximation for TSP. It combines a minimum spanning tree with a minimum matching to obtain a Hamiltonian cycle of the graph. The length of the spanning tree is at most the same as the length of the optimal cycle and the length of the minimum matching can be bounded by one half of the length of the optimal cycle as follows: A perfect matching can be extracted from a Hamiltonian cycle (in a graph with an even number of vertices) by taking every other edge. Of course, there are two such matchings and their costs sum up to exactly the costs of the cycle. This means that there is one matching with at most half of the costs.

This strategy cannot be directly applied to multiobjective optimization, since there, it is not clear what "take the best of some alternatives" means. We show that discrepancy theory provides a tool such that this idea can still be transferred to multiobjective optimization. The Beck-Fiala theorem [BF81] shows how to divide a set of objects with multi-dimensional costs into two alternatives such that the costs are roughly the same in every component. This way, we can always find a solution with about half the total

costs (cf. section 7.2.3). As an example, we apply this result to the maximum traveling salesperson problem. Glaßer et al. [GRW11] also use this technique for the multiobjective maximum satisfiability problem.

We start with an introductory part, then show how to approximate multiobjective maximum cycle covers in graphs, which will be a subroutine in the approximation algorithm for the traveling salesperson problem. Next, we show how Doerr and Srivastav [DS03] extended the Beck-Fiala theorem to more than two alternatives. Finally, we put all parts together and show that for any number of objectives, the maximum traveling salesperson problem is $2/3$-approximable on undirected graphs and $1/2$-approximable on directed graphs.

The Beck-Fiala theorem and also the multi-color extension of Doerr and Srivastav has one drawback: We do not know how the objects are divided into the two (or more) alternatives. Suppose we have a tour in an $\mathbb{N}^k$-labeled graph and want to divide this tour into two sets of edges with roughly the same weight such that each set is well connected. Using the Beck-Fiala theorem, it can happen that we end up with two sets that only contain isolated edges. In the next chapter, we give a result that shows how to obtain such a division that guarantees the same error bound as the Beck-Fiala theorem and at the same time, each of the two sets contains not more than a constant number of connected components. Furthermore, this result can be generalized to more than two alternatives. This result could turn out to be applicable to an even wider range of multiobjective optimization problems.

It should be noted that the $1/2$-approximation for k-MaxSTSP was first obtained without the results from discrepancy theory. An older algorithm used maximum perfect matchings and the result from the next chapter. For that algorithm, it was essential that the number of changes between the two alternatives is bounded by a constant, because the maximum matching needs a certain structure. This problem does not occur when one uses maximum cycle covers, as we will do now.

Definitions. If the traveling salesperson problem is turned into a maximization problem, the goal is to find a Hamiltonian cycle of maximum weight given some complete $\mathbb{N}$-labeled graph as input. In contrast to the minimization problem, we do not require the costs to be metric and thus the generalization to multiple objectives is straightforward. For directed graphs this problem is called k-objective maximum (asymmetric) traveling salesperson (k-MaxATSP), while for undirected graphs it is called k-objective maximum symmetric traveling salesperson (k-MaxSTSP). Since k-MaxATSP allows general directed labeled graphs as input, which includes non-asymmetric labelings, we will drop the word "asymmetric".

Below we repeat the formal definition of k-MaxATSP from section 4.2, the problem k-MaxSTSP is defined analogously.

Definition 7.14 (k-Objective Maximum Traveling Salesperson)**.**

k-MaxATSP $= (S, f, \leq)$ where

- instances are $\mathbb{N}^k$-edge-labeled directed complete graphs $G = (V, E, l)$,
- $S^{\langle G \rangle} = \{C \subseteq E \mid C \text{ is a Hamiltonian cycle of } G\}$ and
- $f^{\langle G \rangle}(C) = \sum_{e \in C} l(e)$.

Previous Work. In 1979, Fisher, Nemhauser and Wolsey [FNW79] gave a $1/2$-approximation algorithm for the single-objective maximum traveling salesperson problem (1-MaxATSP) by removing the lightest edge from each cycle of a maximum cycle cover and connecting the remaining paths to a Hamiltonian cycle. Since undirected cycles always contain at least three edges, this also showed that the single-objective maximum symmetric traveling salesperson problem (1-MaxSTSP) is $2/3$-approximable. We will use this idea and extend it to multiple objectives. Since then, many improvements were achieved, and currently, the best known approximation ratios of $2/3$ for 1-MaxATSP and $7/9$ for 1-MaxSTSP are due to Kaplan et al. [KLSS05] and Paluch, Mucha and Madry [PMM09].

For k-MaxATSP and k-MaxSTSP, where $k \geq 2$, the currently best known approximation algorithms are due to Manthey, who showed a randomized $(1/2 - \varepsilon)$-approximation of k-MaxATSP and a randomized $(2/3 - \varepsilon)$-approximation of k-MaxSTSP [Man09]. Recently, Manthey also showed a deterministic $(1/2k - \varepsilon)$-approximation of k-MaxSTSP and a deterministic $(1/(4k-2) - \varepsilon)$-approximation of k-MaxATSP [Man11].

7.2.2 Approximating Cycle Covers

We will consider approximation algorithms for the multiobjective traveling salesperson problem that use approximation algorithms for multiobjective version of the maximum cycle cover problem as subroutines. For directed input graphs we have the following problem definition.

Definition 7.15 (k-Objective Maximum Directed Edge-Fixed c-Cycle Cover)**.**

k-c-MaxDCC$_{\mathrm{F}} = (S, f, \geq)$ where

- instances consist of a $\mathbb{N}^k$-labeled complete directed graph $G = (V, E, w)$ and a set of edges $F \subseteq E$,
- $S^{\langle G,F\rangle} = \{C \subseteq E \mid C$ is a cycle cover of G with at least c edges per, cycle and $F \subseteq C\}$
- $f^{\langle G,F\rangle}(C) = \sum_{e \in C} w(c)$.

For undirected input graphs we analogously define the k-objective maximum undirected edge-fixed c-cycle cover problem (k-c-MaxUCC$_{\mathrm{F}}$, for short). Let k-c-MaxUCC (k-c-MaxDCC) denote the problems we obtain from k-c-MaxDCC$_{\mathrm{F}}$ (k-c-MaxUCC$_{\mathrm{F}}$) if we require $F = \emptyset$. Using this notation we obtain the usual cycle cover problems k-MaxDCC as k-0-MaxDCC and k-MaxUCC as k-0-MaxUCC (cf. Definition 4.15).

Manthey and Ram [MR09] show by a reduction to matching that there is an FPRAS for k-objective *minimum* cycle cover problems. The same technique can be used to show that there are FPRAS for k-MaxDCC and k-MaxUCC [Man09]. We show that there are FPRAS for k-2-MaxDCC$_{\mathrm{F}}$ and k-3-MaxUCC$_{\mathrm{F}}$ by a reduction to k-MaxDCC and k-MaxUCC.

Theorem 7.16. For every $k \geq 1$, k-2-MaxDCC$_{\mathrm{F}}$ and k-3-MaxUCC$_{\mathrm{F}}$ admit an FPRAS.

Proof. For every $l \geq 1$, let l-`MaxDCC-Approx` (l-`MaxUCC-Approx`) denote the FPRAS for l-MaxDCC (l-MaxUCC). We begin with the directed case.

Let $k \geq 1$. On input of the $\mathbb{N}^k$-labeled complete directed graph $G = (V, E, w)$ and $F \subseteq E$, let $G' = (V, E, w')$, where $w' \colon E \to \mathbb{N}^{k+1}$ such that for all $e \in E$,

$$w'_i(e) = w_i(e) \qquad \text{for } 1 \leq i \leq k$$
$$w'_{k+1}(e) = \begin{cases} 1 & \text{if } e \in F \\ 0 & \text{otherwise.} \end{cases}$$

For $\varepsilon > 0$, apply the algorithm $(k+1)$-`MaxDCC-Approx` to G' with an approximation ratio of $\varepsilon' = \min\{\varepsilon, 1/(r+1)\}$, where $r := \#F$ and return the obtained set of cycle covers that contain all edges from F.

Let C be some (arbitrary) cycle cover with $F \subseteq C$. If no such cycle cover exists, we are done. Otherwise, we have $w'_{k+1}(C) = r$, and with probability at least $1/2$ the FPRAS must have returned some cycle cover C' that ε'-approximates C. By $\varepsilon' \leq 1/(r+1)$ we have $w'_{k+1}(C') \geq (1 - \varepsilon') \cdot w'_{k+1}(C) \geq (1 - 1/(r+1)) \cdot r > r - 1$ and hence $F \subseteq C'$. Moreover, by $\varepsilon' \leq \varepsilon$ we have $w_i(C') = w'_i(C') \geq (1 - \varepsilon') \cdot w'_i(C) \geq (1 - \varepsilon) \cdot w'_i(C) = (1 - \varepsilon) \cdot w_i(C)$ for all $1 \leq i \leq k$. Since an arbitrary cycle in a complete directed graph has length at least two, the assertion is proved.

The proof for the undirected case is similar: We call $(k+1)$-`MaxUCC-Approx` instead and since in a complete undirected graph every cycle has length at least three, the assertion follows. □

7.2.3 Multi-Color Discrepancy

Suppose we have a list of items with (single-objective) weights and want to find a subset of these items with about half of the total weight. The exact version of this problem is of course the NP-complete problem Partition [GJ79], and hence it is unlikely that an exact solution can be found in polynomial time. If we allow a deviation in the order of the largest weight, this problem can be solved in polynomial time, though. Surprisingly, this is still true if the weights are not single numbers but vectors of numbers, which follows from a classical result in discrepancy theory known as the Beck-Fiala theorem [BF81]. It is important to note that the allowed deviation is independent of the number of vectors since this enables us to use this result in multiobjective approximation for balancing out multiple objectives at the same time with an error that does not depend on the input size.

In the Beck-Fiala theorem and the task discussed above, we have to decide for each item to either include it or not. In some situations in multiobjective optimization, though, a more general problem needs to be solved: There is a constant number of weight vectors for each item, out of which we have to choose exactly one.

Doerr and Srivastav [DS03] showed that the Beck-Fiala theorem generalizes to this so-called multi-color setting with almost the same deviation. Their proof implicitly shows that this choice can be computed in polynomial time. For completeness we restate the proof and argue for polynomial-time computability.

Remember that for a vector $x \in \mathbb{Q}^m$, the maximum norm is defined as $\|x\|_\infty = \max_i |x_i|$, and for a matrix $A \in \mathbb{Q}^{m \times n}$ let $\|A\|_1 = \max_j \sum_i |a_{ij}|$. A coloring of n elements with c colors per element can be interpreted as a vector $x \in \{0,1\}^{cn}$ such that for all $b \in \{1, \ldots, n\}$ there is at most one $k_b \in \{0, \ldots, c-1\}$ such that $x_{cb-k_b} = 1$. So the object with index b is colored with k_b. The advantage of this view is that we can also consider *partial*

colorings: A partial coloring is a vector $x \in \mathbb{Q}^{cn}$ such that for all $b \in \{1, \dots, n\}$ it holds that $\sum_{k=0}^{c-1} x_{cb-k} = 1$. So for each object we can assign weights to the colors that have to sum up to one.

For $c \geq 2$, $n \geq 1$ we define the set of all partial colorings of n elements with c colors as $\overline{M_{c,n}} = \{x \in (\mathbb{Q} \cap [0,1])^{cn} \mid \sum_{k=0}^{c-1} x_{cb-k} = 1 \text{ for all } b \in \{1, \dots, n\}\}$ and the set of all real colorings as $M_{c,n} = \overline{M_{c,n}} \cap \{0,1\}^{cn}$.

Theorem 7.17 (Doerr, Srivastav [DS03])**.** There is a polynomial-time algorithm that on input of some $A \in \mathbb{Q}^{m \times cn}$, $m, n \in \mathbb{N}$, $c \geq 2$ and $p \in \overline{M_{c,n}}$ finds a coloring $\chi \in M_{c,n}$ such that $\|A(p-\chi)\|_\infty \leq 2\|A\|_1$.

Proof. Let $\Delta := \|A\|_1$. We start with the partial coloring $\chi = \chi^{(0)} = p \in \overline{M_{c,n}}$ and will successively change it to a vector in $M_{c,n}$. We will first describe the algorithm and then argue about its runtime.

Let $J := J(\chi) := \{j \in \{1, \dots, cn\} \mid \chi_j \notin \{0,1\}\}$ and call the columns from J floating. Let $I := I(\chi) := \{i \in \{1, \dots, m\} \mid \sum_{j \in J(\chi)} |a_{ij}| > 2\Delta\}$. We will ensure that during the rounding process the following conditions are fulfilled (this is clear from the start, because $\chi^{(0)} = p$):

$$(A(p-\chi))_{|I} = 0 \qquad \text{(C1)} \qquad\qquad \chi \in \overline{M_{c,n}} \qquad \text{(C2)}$$

Let us assume that the rounding process is at step t where the current coloring is $\chi = \chi^{(t)}$ and the conditions (C1) and (C2) hold. If there is no floating column, i.e., $J = \emptyset$, then $\chi \in M_{c,n}$ and thus χ has the desired form.

Otherwise, assume that there are still floating columns. Let $B = \{b \in \{1, \dots, n\} \mid \exists k \in \{0, \dots, c-1\} \colon cb - k \in J\}$ be the c-blocks that contain floating columns. Since $\chi \in \overline{M_{c,n}}$, a c-block of χ contains either none or at least two floating columns, thus $\#B \leq \frac{1}{2}\#J$.

Since

$$\#J \cdot \Delta = \sum_{j \in J} \Delta \geq \sum_{j \in J} \sum_{i=1}^{m} |a_{ij}| \geq \sum_{j \in J} \sum_{i \in I} |a_{ij}| = \sum_{i \in I} \sum_{j \in J} |a_{ij}| > \sum_{i \in I} 2\Delta = \#I \cdot 2\Delta$$

it holds that $\#I < \frac{1}{2}\#J$. Consider the inhomogeneous system of linear equations

$$\begin{aligned} &(A(p-\chi))_{|I} = 0 \\ &\sum_{k=0}^{c-1} \chi_{cb-k} = 1 \qquad\qquad \text{for } b \in B \end{aligned}$$

where each χ_j is considered as a variable if $j \in J$ and as a constant if $j \notin J$. This system consists of at most $\#I + \#B < \frac{1}{2}\#J + \frac{1}{2}\#J = \#J$ equations and $\#J$ variables and hence is under-determined. Note that the system has the solution $\chi_{|J}$ because χ fulfills the conditions (C1) and (C2). Since it is under-determined, it also has a second solution $x \in \mathbb{Q}^J$. We extend x to $x_E \in \mathbb{Q}^{cn}$ by

$$(x_E)_j = \begin{cases} x_j & \text{if } j \in J \\ \chi_j & \text{otherwise.} \end{cases}$$

Consider the line $\{(1-\lambda)\,\chi + \lambda\, x_E \mid \lambda \in \mathbb{Q}\}$. Each point on this line (or rather its restriction to the components in J) fulfills the system of equations and thus condition (C1). By condition (C2) and the definition of J it holds that $0 < \chi_j < 1$ for all $j \in J$ and thus there is some $\lambda \in \mathbb{Q}$ such that $\chi^{(t+1)} := (1-\lambda)\,\chi + \lambda\, x_E \in \overline{M_{c,n}}$ and at least one component becomes 0 or 1, i.e., $J(\chi^{(t+1)}) \subsetneq J(\chi^{(t)})$. Note that $\chi^{(t+1)}$ fulfills (C1) and (C2) even for the larger sets $J(\chi^{(t)})$ and $I(\chi^{(t)})$. Continue the rounding process with $\chi := \chi^{(t+1)}$.

Since at least one column is removed from J in each iteration, the rounding process will eventually stop. Let χ be the final value of the coloring. We show $\|A(p-\chi)\|_\infty \leq 2\Delta$. Let $1 \leq i \leq m$. Since at the end, $J = \emptyset$ we also have $I = \emptyset$. Let $\chi^{(t)}$ be the first coloring such that $i \notin I$. Since $\chi^{(t)}$ fulfills (C1) also for $I(\chi^{(t-1)})$ (or $I(\chi^{(0)})$ if $t = 0$) we have $(A(p-\chi^{(t)}))_i = 0$. Furthermore it holds that $\chi_j^{(t)} = \chi_j$ for all $j \notin J(\chi^{(t)})$ and $|\chi_j^{(t)} - \chi_j| < 1$ for all $j \in J(\chi^{(t)})$. Finally note that $\sum_{j \in J(\chi^{(t)})} |a_{ij}| \leq 2\Delta$ since $i \notin I(\chi^{(t)})$. Combining these facts, we obtain

$$|(A(p-\chi))_i| = |(A(p-\chi^{(t)}))_i + (A(\chi^{(t)} - \chi))_i| = |0 + \sum_{j \in J(\chi^{(t)})} a_{ij}(\chi_j^{(t)} - \chi_j)| \leq 2\Delta.$$

We now analyze the runtime. Note that we have at most cn iterations, which is polynomial in the input length. In each iteration, we have to solve an inhomogeneous system of linear equations and we have to find a certain $\lambda \in \mathbb{Q}$. The system, whose size is polynomial in the input length, can be solved in polynomial time (see for instance [GLS88, Theorem 1.4.8]). By adding an equation of the form $\chi_j = 2$ for some suitable $j \in J$, we can find a solution different to χ. The value for λ can be obtained in polynomial time by successively trying to fix each floating column to 0 or 1, solving for λ and checking if the resulting vector is still in $\overline{M_{c,n}}$. □

Corollary 7.18. There is a polynomial-time algorithm that on input of the vectors $v^{j,r} \in \mathbb{Q}^m$ for $1 \leq j \leq n$, $1 \leq r \leq c$ computes a coloring $\chi\colon \{1,\dots,n\} \to \{1,\dots,c\}$ such that for each $1 \leq i \leq m$ it holds that

$$\left| \frac{1}{c} \sum_{j=1}^{n} \sum_{r=1}^{c} v_i^{j,r} - \sum_{j=1}^{n} v_i^{j,\chi(j)} \right| \leq 2m \max_{j,r} |v_i^{j,r}|.$$

Proof. The result is obvious for $c = 1$. For $c \geq 2$, we use Theorem 7.17. Because the error bound is different for each row, we need to scale the rows of the vectors. Let $\delta_i = \max_{j,r} |v_i^{j,r}|$ for $1 \leq i \leq m$. Let $A = (a_{i,j'}) \in \mathbb{Q}^{m \times cn}$ where $a_{i,(c(j-1)+r)} = \frac{1}{\delta_i} v_i^{j,r}$ (if $\delta_i = 0$, set it to 0) and $p \in \mathbb{Q}^{cn}$ such that $p_i = \frac{1}{c}$ for all $1 \leq i \leq cn$. We obtain a coloring $\chi \in \{0,1\}^{cn}$ such that for each $1 \leq j \leq n$ there is exactly one $1 \leq r \leq c$ such that $\chi_{c(j-1)+r} = 1$ and it holds that $\|A(p-\chi)\|_\infty \leq 2\|A\|_1$. Note that because of the scaling, the largest entry in A is 1 and thus we have $\|A\|_1 \leq m$. Define $\chi'\colon \{1,\dots,n\} \to \{1,\dots,c\}$ by $\chi'(j) = r \iff \chi_{c(j-1)+r} = 1$. We obtain for each $1 \leq i \leq m$:

$$2m\delta_i \geq 2\|\delta_i\, A\|_1 \geq |(\delta_i\, A(p-\chi))_i| = \left| \sum_{j'=1}^{cn} \delta_i a_{ij'}(p_{j'} - \chi_{j'}) \right| = \left| \sum_{j=1}^{n} \sum_{r=1}^{c} \frac{1}{c} v_i^{j,r} - \sum_{j=1}^{n} v_i^{j,\chi'(j)} \right|$$

□

7.2.4 Approximating Multiobjective Maximum Traveling Salesperson

We show that k-MaxATSP is randomized $1/2$-approximable and k-MaxSTSP is randomized $2/3$-approximable using the following idea. We choose a suitable number l depending only on k and try all sets of at most l edges F using brute force. For each such F we apply the FPRAS for k-2-MaxDCC$_\mathrm{F}$ (k-3-MaxUCC$_\mathrm{F}$), which exists by Theorem 7.16, fixing the edges in F. For all cycle covers thus obtained, we select two (three) edges from each cycle and compute a 2-coloring (3-coloring) of the cycles with low discrepancy with regard to the weight vectors of the selected edges. Using this coloring, we remove exactly one edge from each cycle and connect the remaining paths to a single cycle in an arbitrary way. Since the coloring has low discrepancy, we only remove about one half (one third) of the weight in each objective. The introduced error is absorbed by choosing suitable heavy edges F at the beginning. The described procedure generally works for arbitrary c-cycle covers.

Algorithm 7.4: k-`MaxTSP-Approx`(V, E, w) with parameter $c \geq 2$

`Input` : $\mathbb{N}^k$-labeled directed/undirected complete graph $G = (V, E, w)$
`Output`: set of Hamiltonian cycles of G

1 `foreach` $F_H, F_L \subseteq E$ with $\#F_H \leq 3\,c\,k^2$, $\#F_L \leq c\#F_H$: `do`
2 let $\delta \in \mathbb{N}^k$ with $\delta_i = \max\{n \in \mathbb{N} \mid$ there are $3\,c\,k$ edges $e \in F_H$ with $w_i(e) \geq n\}$;
3 `foreach` $e \in E - F_H$ `do`
4 `if` $w(e) \not\leq \delta$ `then` set $w(e) = 0$ for the current iteration of the loop in line 1;
5 compute a $(1 - 1/\#V)$-approximation $\mathcal{S}$ of k-c-MaxDCC$_\mathrm{F}$ / k-c-MaxUCC$_\mathrm{F}$ on input of $(G, F_H \cup F_L)$;
6 `foreach` cycle cover $S \in \mathcal{S}$ `do`
7 let $C_1, \ldots, C_r$ denote the cycles in S;
8 `if` for each $i \in \{1, \ldots, r\}$, $C_i - F_H$ contains a path of length c `then`
9 `foreach` $i \in \{1, \ldots, r\}$ `do` choose path $e_{i,1}, \ldots, e_{i,c} \in C_i - F_H$ arbitrarily;
10 compute some coloring $\chi\colon \{1, \ldots, r\} \to \{1, \ldots, c\}$ such that

$$\sum_{i=1}^{r} w(e_{i,\chi(i)}) \quad \leq \quad 2k \cdot \delta + \frac{1}{c} \sum_{i=1}^{r} \sum_{j=1}^{c} w(e_{i,j})$$

  and remove the edges $\{e_{i,\chi(i)} \mid i = 1, \ldots, r\}$ from S;
11 output the remaining edges, arbitrarily connected to a Hamiltonian cycle;

Lemma 7.19. Let $c \geq 2$ and $k \geq 1$. If there exists an FPRAS for k-c-MaxDCC$_\mathrm{F}$ (k-c-MaxUCC$_\mathrm{F}$, resp.), then Algorithm 7.4 computes a randomized $(1 - 1/c)$-approximation for k-MaxATSP (k-MaxSTSP, resp.).

Proof. Let $k \geq 1$, $c \geq 2$, and $G = (V, E, w)$ be some $\mathbb{N}^k$-labeled (directed or undirected) input graph with $m = \#V$ sufficiently large.

We will first argue that the algorithm terminates in time polynomial in the length of G. Since there are only polynomially many subsets $F_H, F_L \subseteq E$ with cardinality bounded by a constant, the loop in line 1 is executed polynomially often. In each iteration the FPRAS on $G = (V, E, w)$ and $F_H \cup F_L \subseteq E$ terminates in time polynomial in the length

of G and $F_H \cup F_E$, which means that the set $\mathcal{S}$ contains only polynomially many cycle covers. Hence, for each iteration of the loop in line 1, the loop in line 6 is also executed at most polynomially many times, and overall we have polynomially many nested iterations. In each nested iteration where each cycle of the cycle cover contains a path as required, we compute a coloring of $\{1, \dots, r\}$ with low discrepancy. By Corollary 7.18 this can be done in polynomial time. Observe that all further steps require at most polynomial time, and hence the algorithm terminates after polynomially many steps.

Next we argue that the algorithm will succeed with probability at least $1/2$. Observe that the only randomized parts of the algorithm are the calls to the randomized cycle cover approximation algorithm in line 5. Using amplification we can assume that the probability that all the calls to this algorithm succeed is at least $1/2$.

It remains to show that if Algorithm 7.4 succeeds, it outputs some $(1-1/c)$-approximate set of Hamiltonian cycles. Hence, for the remainder of the proof, let us assume that the algorithm and hence all calls to the internal FPRAS succeed. Furthermore, let $R \subseteq E$ be some Hamiltonian cycle of G. We will argue that there is some iteration where the algorithm outputs an $(1 - 1/c)$-approximation of R.

For each $1 \leq i \leq k$, let $F_{H,i} \subseteq R$ be some set of $3\,c\,k$ heaviest edges of R in the i-th component, breaking ties arbitrarily. Let $F_H = \bigcup_{i=1}^{k} F_{H,i}$. We define $F_L \subseteq R$ such that $F_L \cap F_H = \emptyset$ and each edge in F_H is part of a path in $F_L \cup F_H$ that contains c edges from F_L. This is always possible as long as R is large enough. We now have $\#F_H \leq 3\,c\,k^2$ and $\#F_L \leq c\,\#F_H$. Hence in line 1 there will be some iteration that chooses F_H and F_L. We fix this iteration for the remainder of the proof.

Let $\delta \in \mathbb{N}^k$ as defined in line 2 and observe that $\delta_i = \min\{w_i(e) \mid e \in F_{H,i}\}$ for all i, which means that for all edges $e \in R - F_H$ we have $w(e) \leq \delta$. Hence the loop in line 3 sets the weights of all edges $e \in E - R$ that do not fulfill $w(e) \leq \delta$ to zero, and these are the only weights that are modified. In particular, this does not affect edges in R, hence $w(R)$ remains unchanged. Note that since we do not increase the weight of any edge and do not change the weight of the edges in R, it suffices to show that the algorithm computes an approximation with respect to the changed weights.

Next we obtain a $(1 - 1/\#V)$-approximate set $\mathcal{S}$ of c-cycle covers of G that contain $F_H \cup F_L$. Since R is a c-cycle cover of G with $F_H \cup F_L \subseteq R$, there must be some c-cycle cover $S \in \mathcal{S}$ with $F_H \cup F_L \subseteq S$ that $(1 - 1/\#V)$-approximates R. Hence in line 6 there will be some iteration that chooses this S. Again we fix this iteration for the remainder of the proof.

As in line 7, let $C_1, \dots, C_r$ denote the cycles in S. Note that each cycle contains at least c edges. Since each edge in F_H is part of a path in $F_H \cup F_L$ with at least c edges from F_L, we even know that each cycle contains at least c edges not from F_H and thus the condition in line 8 is fulfilled. Let these edges $e_{i,j}$ be defined as in the algorithm. Note that since $e_{i,j} \notin F_H$ we have $w(e_{i,j}) \leq \delta$ for all i, j, because the weight function was changed accordingly.

In line 10 we compute some $\chi\colon \{1, \dots, r\} \to \{1, \dots, c\}$ such that

$$\begin{aligned} \sum_{i=1}^{r} w(e_{i,\chi(i)}) &\leq 2\,k \cdot \delta + \frac{1}{c} \sum_{i=1}^{r} \sum_{j=1}^{c} w(e_{i,j}) \\ &\leq 2\,k \cdot \delta + \frac{1}{c} \cdot w(S - F_H). \end{aligned}$$

Recall that by Corollary 7.18 such a coloring exists and can be computed in polynomial time. Removing the chosen edges breaks the cycles into paths, which can be arbitrarily connected to a Hamiltonian cycle R'. For the following estimation note that $\delta \leq \frac{w(F_H)}{3\,c\,k}$ and $w(F_H) \geq \frac{3\,c\,k}{m} w(R)$ and recall that $m = \#V = \#R$.

$$\begin{aligned}
w(R') &\geq w(S) - \sum_{i=1}^{r} w(e_{i,\chi(i)}) \\
&\geq w(S) - 2\,k \cdot \delta - \frac{1}{c} \cdot w(S - F_H) \\
&= \left(1 - \frac{1}{c}\right) w(S) + \frac{1}{c} w(F_H) - 2\,k \cdot \delta \\
&\geq \left(1 - \frac{1}{c}\right) w(S) + \frac{1}{3\,c} w(F_H) \\
&\geq \left(1 - \frac{1}{c}\right) \left(1 - \frac{1}{m}\right) w(R) + \frac{k}{m} w(R) \\
&= \left(1 - \frac{1}{c}\right) w(R) + \left(-\left(1 - \frac{1}{c}\right) + k\right) \frac{1}{m} w(R) \\
&\geq \left(1 - \frac{1}{c}\right) w(R)
\end{aligned}$$

This proves the assertion. □

It is known that 1-c-MaxDCC is APX-hard for all $c \geq 3$ [BM05] and that 1-c-MaxUCC is APX-hard for $c \geq 5$ [Man08]. This means that, unless P = NP, there is no PTAS for these problems (and especially not for the variants with fixed edges). Furthermore, the existence of an FPRAS or PRAS for these problems implies NP = RP and thus a collapse of the polynomial-time hierarchy, which is seen as follows.

If an APX-hard problem has a PRAS, then all problems in APX have a PRAS and hence Max3Sat has one. There exists an $\varepsilon > 0$ and a polynomial-time computable f mapping CNF formulas to 3-CNF formulas such that if $x \in$ Sat, then $f(x) \in$ 3Sat; and if $x \notin$ Sat, then there is no assignment satisfying more than a fraction of $1 - \varepsilon$ of the clauses in $f(x)$ [AL97, Theorem 10.1]. The PRAS for Max3Sat allows us to compute probabilistically a $(1 - \varepsilon/2)$-approximation for $f(x)$ which in turn tells us whether or not $x \in$ Sat. Since this procedure has no false negatives we get RP = NP, which implies a collapse of the polynomial-time hierarchy [Lau83, Sip83].

So it seems unlikely that there is a PRAS for 1-c-MaxDCC where $c \geq 3$ and 1-c-MaxUCC where $c \geq 5$. However, this does not necessarily mean that the above algorithm is useless for parameters $c \geq 3$ in the directed and $c \geq 5$ in the undirected case: The algorithm could still benefit from a constant-factor approximation for k-c-MaxUCC$_{\mathrm{F}}$ or k-c-MaxDCC$_{\mathrm{F}}$. A simple change in the estimation shows that if the cycle cover algorithm has an approximation ratio of α, the above algorithm provides an approximation with ratio $\alpha(1 - 1/c)$.

Theorem 7.20. Let $k \geq 1$.

1. k-MaxATSP is randomized 1/2-approximable.
2. k-MaxSTSP is randomized 2/3-approximable.

Proof. We combine Theorem 7.16 and Lemma 7.19. □

Chapter 8

Discrepancy Theory

Discrepancy theory tries to approximate continuous distributions by discrete ones and the error that occurs therein is generally called discrepancy. For example, one problem in geometric discrepancy theory is to place n points in the unit square such that for each rectangle R in the unit square, the ratio of points that lie in R is as close as possible to the area of R. Another topic in discrepancy theory are rounding problems. Here, one for example has a list of real numbers and tries to round them to integers while trying to change their sum as little as possible. The classical theorem of Beck and Fiala [BF81] belongs to this area of discrepancy theory. It provides an upper bound for certain weighted roundings.

Theorem 8.1 (Beck, Fiala [BF81])**.** For any $y \in \mathbb{R}^n$ and any weighting matrix $A \in \mathbb{R}^{m \times n}$ there is a rounding $x \in \mathbb{Z}^n$ of y such that

$$\|Ay - Ax\|_\infty \leq \|A\|_1.$$

Furthermore, if all inputs are rational, this rounding can be found in polynomial time.

It must be noted that the original formulation of this theorem was less general, but the proof can also be applied to this setting. We call $x \in \mathbb{Z}^n$ a rounding of $y \in \mathbb{R}^n$ if $|x_i - y_i| < 1$ for all i, i. e. $\|x - y\|_\infty < 1$. Remember that $\|x\|_\infty = \max_i |x_i|$ and $\|A\|_1 = \max_j \sum_i |a_{ij}|$, so the upper bound does not depend on n, the number of components of x.

The main contribution of this chapter is to show that these roundings can have a certain structure (Theorem 8.18): We show that if all components of $y \in \mathbb{R}^n$ are equal, there is a rounding x of y such that the number of changes in the direction of the rounding does not depend on n, while the error bound remains the same as in the theorem by Beck and Fiala. Doerr and Srivastav extended the Beck-Fiala theorem to multiple colors. We note that such an extension is also possible here but will not be covered.

The methods to achieve this result will be analytic. Using topological degree theory we show a multi-dimensional intermediate value theorem for integrals. One can formulate the usual intermediate value theorem for integrals as follows: For any integrable function $f\colon [0,1] \to \mathbb{R}$ and any $0 \leq \alpha \leq 1$, there is an interval $[0, x] \subseteq [0, 1]$ such that $\int_{[0,x]} f(x)\,dx = \alpha \int_{[0,1]} f(x)\,dx$. We extend this to multiple dimensions as follows: For any componentwise integrable function $f\colon [0,1] \to \mathbb{R}^n$ and any $0 \leq \alpha \leq 1$, we can find some $X \subseteq [0,1]$ such that for each i, $\int_X f_i(x)\,dx = \alpha \int_{[0,1]} f_i(x)\,dx$ and furthermore, X is the union of at most $2n$ intervals. We get this result by first showing a multi-dimensional mean value theorem.

We can of course apply the mean value theorem again on the set of intervals obtained and thus reach a ratio of $1/4$. The main difficulty here lies in keeping the number of needed intervals small enough. By iteratively applying this idea, we can reach every ratio of the form $r/2^k$ and thus get the final result by a compactness argument. We also show that the number of needed intervals for the intermediate value theorem is tight.

Preliminaries. Let $X \subseteq \mathbb{R}^m$ be bounded. We call a function $f\colon X \to \mathbb{R}$ *integrable*, if it is Lebesgue-integrable on X. This is especially the case if f is bounded and has only finitely many points of discontinuity. A function $g\colon X \to \mathbb{R}^n$ is *componentwise integrable*, if all projections g_i are integrable and in this case we write $\int_X g(x)\,dx$ as abbreviation for the vector $(\int_X g_1(x)\,dx, \dots, \int_X g_n(x)\,dx)^{\mathrm{T}}$.

For a set $A \subseteq \mathbb{R}^n$, we write ∂A for its *boundary* and, contrary to the notation used in other chapters, $\overline{A}$ denotes the *(topological) closure* of A.

8.1 A Multi-Dimensional Mean Value Theorem

One can use the mean value theorem to show that for any integrable $f\colon [0,1] \to \mathbb{R}$ there is some $x \in [0,1]$ such that

$$\int_0^x f(x)\,dx = \frac{1}{2}\int_0^1 f(x)\,dx.$$

This is no longer true in general if f maps to $\mathbb{R}^n$ for $n \geq 2$. Take, for example, $f(x) = (1,0)^{\mathrm{T}}$ for $x \leq \frac{1}{2}$ and $f(x) = (0,1)^{\mathrm{T}}$ for $x > \frac{1}{2}$. We can still get a similar result if we allow more degrees of freedom in the domain of integration. In Theorem 8.8 we show that for any integrable $f\colon [0,1] \to \mathbb{R}^{2n}$ there are n intervals $I_1, \dots, I_n \subseteq [0,1]$ such that for $I = I_1 \cup I_2 \cup \dots \cup I_n$,

$$\int_I f(x)\,dx = \frac{1}{2}\int_0^1 f(x)\,dx.$$

The number of needed intervals can be explained by the following argumentation: For a function $f\colon [0,1] \to \mathbb{R}$, it suffices to vary one end of the interval, which corresponds to one degree of freedom. For a function $f\colon [0,1] \to \mathbb{R}^2$, we have seen above that one degree of freedom is not enough. So if we have a function $f\colon [0,1] \to \mathbb{R}^{2n}$, it suffices to have n intervals, which have $2n$ end points and thus $2n$ degrees of freedom. We will see later in Proposition 8.13 that if we change the factor from $1/2$ to an arbitrary ratio between 0 and 1 (and thus arrive at a multi-dimensional *intermediate* value theorem), we need more intervals.

We use a result from topological degree theory to show this multi-dimensional mean value theorem. More specifically, we want to show that a certain function $f\colon D \to \mathbb{R}^n$ has a zero where $D \subseteq \mathbb{R}^n$ is open and bounded. For this, it suffices to show that its *Brouwer degree* at the point $(0, \dots, 0)^{\mathrm{T}}$ is nonzero. Intuitively, the Brouwer degree of a mapping at a point p of its codomain counts the number of preimages of p, where preimages whose orientation is "reversed" by the mapping are counted negatively. Thus, if a point has nonzero Brouwer degree, than it occurs as a value of the mapping (note that the converse does not hold in general). For $n = 2$, the Brouwer degree is similar to the winding number

in complex analysis. We will only give the definition for a special case and refer to the monograph by Lloyd [Llo78] for more details.

Definition 8.2. Let $D \subseteq \mathbb{R}^n$ be open and bounded, $f\colon \overline{D} \to \mathbb{R}^n$ be continuous and $p \in \mathbb{R}^n - f(\partial D)$. Then $\deg(f, D, p)$ denotes the *Brouwer degree* of f at p. If for every $x \in D$ such that $f(x) = p$ it holds that f is (totally) differentiable at x and $\mathrm{D}f(x)$ is non-singular, then we have

$$\deg(f, D, p) = \sum_{\substack{x \in D \\ f(x)=p}} \operatorname{sgn}(\det(\mathrm{D}f(x))).$$

Remember the definition of the *total differential* or the *Jacobian* as

$$\mathrm{D}f(x) = \left(\frac{\partial f_i}{\partial x_j}(x)\right)_{i,j}.$$

The determinant of the total differential reflects the degree of expansion or compression at the point x, depending on its absolute value. Furthermore, its sign shows if the function reverses the "direction" of x and thus the above definition captures the intuitive meaning. We now cite the main results we will use from the book by Lloyd [Llo78].

Theorem 8.3 ([Llo78, Theorem 2.1.1])**.** If $D \subseteq \mathbb{R}^n$ is bounded and open, $f\colon \overline{D} \to \mathbb{R}^n$ is continuous, $p \notin f(\partial D)$, and $\deg(f, D, p) \neq 0$, then $p \in f(D)$.

A set $A \subseteq \mathbb{R}^n$ is called *symmetric* if $x \in A \iff -x \in A$ for all $x \in \mathbb{R}^n$.

Theorem 8.4 (Odd Mapping Theorem, [Llo78, Theorem 3.2.6])**.** Let D be a bounded, open, symmetric subset of $\mathbb{R}^n$ containing the origin. If $f\colon \overline{D} \to \mathbb{R}^n$ is continuous, $0 \notin f(\partial D)$, and for all $x \in \partial D$ it holds that $\frac{f(x)}{|f(x)|} \neq \frac{f(-x)}{|f(-x)|}$, then $\deg(f, D, 0)$ is an odd number (and in particular not zero).

Corollary 8.5. Let D be a bounded, open, symmetric subset of $\mathbb{R}^n$ containing the origin. If $f\colon \overline{D} \to \mathbb{R}^n$ is continuous and for all $x \in \partial D$ it holds that $f(-x) = -f(x)$, then $0 \in f(\overline{D})$.

Proof. Assume that $0 \notin f(\overline{D})$. From $f(-x) = -f(x)$ for $x \in \partial D$ it follows that the inequality condition of Theorem 8.4 is fulfilled (note that $0 \notin f(\partial D)$) and thus $\deg(f, D, 0) \neq 0$ and by Theorem 8.3, $0 \in f(\overline{D})$. This is a contradiction. □

In the following, we introduce some notation that is helpful when talking about unions of intervals. The usefulness of some parts of the definition will become clear in the next section, when we extend the multi-dimensional mean value theorem to a multi-dimensional intermediate value theorem.

Definition 8.6. Define the set of all left closed and right open intervals in $[0, 1)$ and their complements as

$$\begin{aligned}\mathcal{I} = &\{[l, r) \mid 0 \leq l < r \leq 1\} \cup \\ &\{[0, r) \cup [l, 1) \mid 0 < r < l < 1\} \cup \\ &\{\emptyset\}.\end{aligned}$$

For $I \in \mathcal{I}$ with $I \neq \emptyset$ and $I \neq [0,1)$, the points l and r in the definition above are called *left endpoint* and *right endpoint* of I, respectively. For $n \geq 1$, let

$$\mathcal{I}_n = \{I_1 \cup I_2 \cup \cdots \cup I_n \mid I_1, \ldots, I_n \in \mathcal{I}\}$$

be the set of all unions of (at most) n sets from $\mathcal{I}$.

We include "intervals" of the type $[0,r) \cup [l,1)$ in $\mathcal{I}$, because we would like $\mathcal{I}$ to be closed under complementation. In general, the intuition is that zero and one are identified and sets from $\mathcal{I}$ are thus intervals on a circle.

Property 8.7. Let $n \geq 1$. For any $I \in \mathcal{I}_n$, it holds that $[0,1) - I \in \mathcal{I}_n$.

We now have enough terminology to state and prove the main result of this section.

Theorem 8.8. Let $n \geq 1$ and $h\colon [0,1) \to \mathbb{R}^{2n}$ be componentwise integrable. There is some $I \in \mathcal{I}_n$ such that

$$\int_I h(x)\,dx = \int_{[0,1)-I} h(x)\,dx = \frac{1}{2}\int_{[0,1)} h(x)\,dx.$$

Proof. Define $T = \{t \in \mathbb{R}^{2n} \mid \|t\|_1 \leq 1\}$ and for every $t = (t_1, \ldots, t_{2n}) \in T$, let

$$I_t = \bigcup_{\substack{1 \leq k \leq 2n,\\ t_k > 0}} \left[\sum_{i=1}^{k-1} |t_i|, \sum_{i=1}^{k} |t_i|\right)$$

and

$$f\colon T \to \mathbb{R}^{2n}, \quad f(t) = \int_{I_t} h(x)\,dx - \int_{[0,1)-I_t} h(x)\,dx.$$

By the formal definition, I_t is a union of (at most) $2n$ intervals from $\mathcal{I}$. However, it can always be written as a union of at most n intervals by merging adjacent intervals and hence, $I_t \in \mathcal{I}_n$ for all $t \in T$.

We now want to show that $0 \in f(T)$ by applying Corollary 8.5 to f and D being the interior of T. The set D is obviously a bounded, open, and symmetric subset of $\mathbb{R}^{2n}$ containing the origin. The function f is continuous because of the fundamental theorem of calculus for the Lebesgue integral and the fact that the endpoints of the intervals in I_t depend continuously on t. Note that for any $t \in \partial D$, $I_{-t} = [0,1) - I_t$ and thus $f(-t) = -f(t)$. Since all preconditions of the corollary are fulfilled, we get $0 \in f(T)$ and thus there exists some $t \in T$ such that $\int_{I_t} h(x)\,dx = \int_{[0,1)-I_t} h(x)\,dx$. □

8.2 A Multi-Dimensional Intermediate Value Theorem

In Theorem 8.8 it was shown that the integral of a multi-dimensional function can be balanced to exactly half of its total value. In this section, we want to generalize this to arbitrary ratios, i. e. extend this multi-dimensional mean value theorem to a multi-dimensional intermediate value theorem. Note that this does not directly follow from the

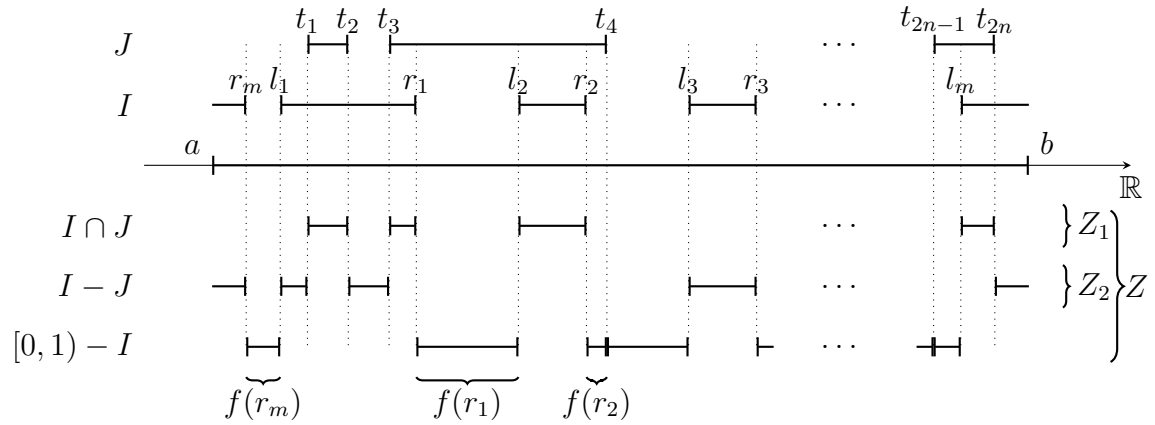


Figure 8.1: Illustration for the construction in the proof of Lemma 8.9. The sets I and J are unions of m resp. n nonempty and pairwise disjoint sets in $\mathcal{I}$. Given all endpoints of I and J, there is a set Z of at most $2m + 2n$ nonempty and pairwise disjoint sets from $\mathcal{I}$ that contains a partition Z_1 of $I \cap J$ and a partition Z_2 of $I - J$. Moreover, all r_i map to disjoint sets $f(r_i) \subseteq [0,1) - I$, hence their removal from Z preserves the partitions Z_1 and Z_2.

continuity of the Lebesgue integral and the (ordinary) intermediate value theorem since we are dealing with multi-dimensional functions. At the end of this section we also show that the number of intervals needed is tight.

We begin by showing that when iteratively intersecting certain sets of intervals, the number of intervals needed to represent the intersection or its complement is bounded.

Lemma 8.9. For $n \geq 1$, let $I \in \mathcal{I}_{2n-1}$, $J \in \mathcal{I}_n$. Then $I \cap J \in \mathcal{I}_{2n-1}$ or $I - J \in \mathcal{I}_{2n-1}$.

Proof. If I or J is empty or I or J is the full interval $[0,1)$, then the lemma is obvious. In all other cases, J contains infinitely many points (i. e. at least one nonempty interval), so we may assume that J is a union of *exactly* n nonempty and pairwise disjoint elements of $\mathcal{I}$ (if not, just split some proper interval in an arbitrary way).

Let $I_1, \ldots, I_m \in \mathcal{I}$, $m \leq 2n-1$ such that $I = I_1 \cup I_2 \cup \cdots \cup I_m$, all I_i are pairwise disjoint and nonempty and no two sets I_i are directly adjacent (not even when identifying 0 and 1). Let $l_1, \ldots, l_m$ and $r_1, \ldots, r_m$ be the left resp. right endpoints of these sets. Because of the above properties, these points are all distinct. Let $t_1, \ldots, t_{2n}$ be the endpoints of the n sets whose union is J (those need not be all distinct). Let $X = \{l_1, \ldots, l_m, r_1, \ldots, r_m, t_1, \ldots, t_{2n}\}$ and sort X in increasing order to obtain the sequence $(x_1, \ldots, x_{\#X})$. Next we define the function $f\colon X \to \mathcal{I}$ by

$$f(x_i) = \begin{cases} [x_i, x_{i+1}) & \text{if } i \neq \#X, \text{ and} \\ [0,1) - [x_1, x_{\#X}) & \text{otherwise.} \end{cases}$$

Clearly, $Z := \{f(x) \mid x \in X\}$ partitions $[0,1)$ into nonempty sets from $\mathcal{I}$. Furthermore, there are sets $Z_1, Z_2 \subseteq Z$ such that $\bigcup_{K \in Z_1} K = I \cap J$ and $\bigcup_{K \in Z_2} K = I - J$ (see Figure 8.1 for an example). Next, we show that $\#Z_1$ or $\#Z_2$ is also small enough.

Due to the fact that no two sets defining I are directly adjacent, we have $f(r_i) \cap I = \emptyset$ for all $1 \leq i \leq m$. Hence, the set $Z - \{f(r_i) \mid 1 \leq i \leq m\}$ still contains Z_1 and Z_2. Since

all r_i are pairwise distinct and f is injective, we have

$$\begin{aligned}\#(Z - \{f(r_i) \mid 1 \leq i \leq m\}) &= \#X - \#\{r_i \mid 1 \leq i \leq m\} \\ &\leq 2m + 2n - m \\ &= 4n - 1.\end{aligned}$$

Observe that Z_1 and Z_2 are disjoint. Hence, one of them must have a cardinality of at most $\lfloor \frac{1}{2}(4n-1) \rfloor \leq 2n - 1$, which proves the assertion. □

We now restate Theorem 8.8 so that it can be applied to split arbitrary unions of intervals as opposed to only the interval $[0, 1)$.

Corollary 8.10. Let $m, n \geq 1$ and $f\colon [0,1) \to \mathbb{R}^{2n}$ be componentwise integrable. For any $I \in \mathcal{I}_m$, there is some $J \in \mathcal{I}_n$ such that

$$\int_{I \cap J} f(x)\,dx = \int_{I - J} f(x)\,dx = \frac{1}{2}\int_I f(x)\,dx.$$

Proof. Let $h\colon [0,1) \to \mathbb{R}^{2n}$ with

$$h(x) = \begin{cases} f(x) & \text{if } x \in I \\ 0 & \text{otherwise.} \end{cases}$$

Clearly, h is componentwise integrable. Hence, by Theorem 8.8 there is some $J \in \mathcal{I}_n$ such that

$$\int_J h(x)\,dx = \int_{[0,1) - J} h(x)\,dx = \frac{1}{2}\int_{[0,1)} h(x)\,dx.$$

By the definition of h, this value is the same as

$$\int_{I \cap J} f(x)\,dx = \int_{I - J} f(x)\,dx = \frac{1}{2}\int_I f(x)\,dx.$$ □

Lemma 8.11. Let $n \geq 1$ and $f\colon [0,1) \to \mathbb{R}^{2n}$ be componentwise integrable. For every $k \in \mathbb{N}$ and every $p \in \{0, 1, \ldots, 2^k\}$ there is some $I \in \mathcal{I}_{2n-1}$ such that

$$\int_I f(x)\,dx = \frac{p}{2^k}\int_{[0,1)} f(x)\,dx.$$

Proof. We show this by induction over k.

Induction Basis ($k = 0$): The assertion is obvious for $k = 0$.

Induction Step ($k \to k+1$): Assuming the validity of the assertion for some k, we first show it for $k+1$ and $p \in \{0, 1, \ldots, 2^k\}$ and subsequently for $k+1$ and $p \in \{2^k + 1, \ldots, 2^{k+1}\}$.

Let $p \in \{0, 1, \ldots, 2^k\}$. By the induction hypothesis, there is some $I \in \mathcal{I}_{2n-1}$ such that

$$\int_I f(x)\,dx = \frac{p}{2^k}\int_{[0,1)} f(x)\,dx.$$

Applying Corollary 8.10 to f and I yields some $J \in \mathcal{I}_n$ such that

$$\int_{I \cap J} f(x)\,dx = \int_{I-J} f(x)\,dx = \frac{1}{2}\int_I f(x)\,dx = \frac{p}{2^{k+1}}\int_{[0,1)} f(x)\,dx.$$

Furthermore, by Lemma 8.9, at least one of $I \cap J$ or $I - J$ can be expressed as a union of $2n-1$ sets in $\mathcal{I}$.

Let now $p \in \{2^k+1, \ldots, 2^{k+1}\}$ and $K \in \mathcal{I}_{2n-1}$ such that

$$\int_K f(x)\,dx = \frac{2^{k+1}-p}{2^{k+1}}\int_{[0,1)} f(x)\,dx,$$

which exists by the first part of the induction step. By Property 8.7, $[0,1)-K \in \mathcal{I}_{2n-1}$ and thus because of

$$\begin{aligned}\int_{[0,1)-K} f(x)\,dx &= \int_{[0,1)} f(x)\,dx - \int_K f(x)\,dx \\ &= \left(1 - \frac{2^{k+1}-p}{2^{k+1}}\right)\int_{[0,1)} f(x)\,dx = \frac{p}{2^{k+1}}\int_{[0,1)} f(x)\,dx\end{aligned}$$

also the second part of the induction step is proved. □

Because of continuity and the fact that $\{\frac{p}{2^k} \mid k \in \mathbb{N} \text{ and } 0 \le p \le 2^k\}$ is dense in $[0,1]$, the previous result can be generalized to arbitrary real factors and thus we obtain a multi-dimensional intermediate value theorem. The main obstacle here is that we need to find a suitable topology on $\mathcal{I}$.

Theorem 8.12. Let $n \ge 1$ and $f\colon [0,1) \to \mathbb{R}^{2n}$ be componentwise integrable. For every real number $\alpha \in [0,1]$ there is some $I \in \mathcal{I}_{2n-1}$ such that

$$\int_I f(x)\,dx = \alpha \int_{[0,1)} f(x)\,dx.$$

Proof. Let $\alpha \in [0,1]$. There is some sequence $\alpha_1, \alpha_2, \ldots \in \{\frac{p}{2^k} \mid k \in \mathbb{N} \text{ and } 0 \le p \le 2^k\}$ converging to α. From Lemma 8.11 we get a sequence $I_1, I_2, \ldots \in \mathcal{I}_{2n-1}$, such that for all i we have

$$\int_{I_i} f(x)\,dx = \alpha_i \int_{[0,1)} f(x)\,dx. \tag{8.1}$$

We now want to show that the sequence $I_1, I_2, \ldots$ has a limit point $\tilde{I} \in \mathcal{I}_{2n-1}$ such that $\int_{\tilde{I}} f(x)\,dx = \alpha \int_{[0,1)} f(x)\,dx$. For this, we need a topology on $\mathcal{I}_{2n-1}$, which we get by indirectly embedding it in Euclidean space. First, note that we may assume that for each i, I_i is a tuple of sets from $\mathcal{I}$, i. e. $I_i \in \mathcal{I}^{2n-1}$. Denote by $\mathrm{S}^2 = \{x \in \mathbb{R}^3 \mid \|x\|_2 = 1\}$ the 2-sphere embedded in $\mathbb{R}^3$. We define a mapping $p\colon \mathcal{I} \to \mathrm{S}^2$ (cf. Figure 8.2): Let $p(\emptyset) := (0,0,-1)^{\mathrm{T}}$, $p([0,1)) := (0,0,1)^{\mathrm{T}}$ and for any $I \in \mathcal{I}$ with left and right endpoints l, r and length $y = \lambda(I) = \int_I 1\,dx = r - l \bmod 1$ let $p(I) := (\cos\theta\cos\varphi, \cos\theta\sin\varphi, \sin\theta)^{\mathrm{T}}$

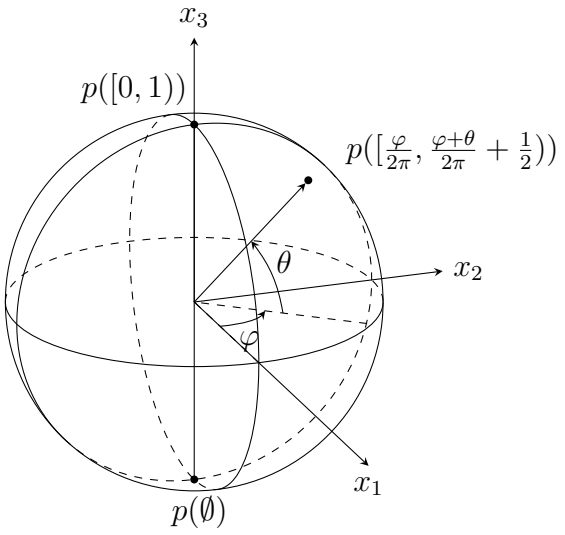


Figure 8.2: Illustration of the bijection $p\colon \mathcal{I} \to \mathrm{S}^2$ that is used to transfer the topology of the 2-sphere S^2 to $\mathcal{I}$ in the proof of Theorem 8.12.

where $\theta = 2\pi(y - \frac{1}{2})$ and $\varphi = 2\pi l$. Note that this mapping is a bijection. We extend it in the obvious way to $p\colon \mathcal{I}^{2n-1} \to (\mathrm{S}^2)^{2n-1}$.

Observe that the function

$$g\colon (\mathrm{S}^2)^{2n-1} \to \mathbb{R}^{2n}, \quad \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_{2n-1} \end{pmatrix} \mapsto \int\limits_{\bigcup_{k=1}^{2n-1} p^{-1}(a_k)} f(x)\,dx$$

is continuous (with respect to the relative topology of $(\mathrm{S}^2)^{2n-1} \subseteq \mathbb{R}^{6n-3}$) and that for any i, it holds that

$$\int\limits_{I_i} f(x)\,dx = g(p(I_i)). \tag{8.2}$$

Since $(\mathrm{S}^2)^{2n-1}$ is a compact subset of $\mathbb{R}^{6n-3}$, the sequence $p(I_1), p(I_2), \dots \in (\mathrm{S}^2)^{2n-1}$ has a convergent subsequence $p(I_1'), p(I_2'), \dots$ such that $\lim_{j\to\infty} p(I_j') =: \tilde{a} \in (\mathrm{S}^2)^{2n-1}$. Let $\tilde{I} = \bigcup_{k=1}^{2n-1} p^{-1}(\tilde{a}_k) \in \mathcal{I}_{2n-1}$. We finally have

$$\begin{aligned}\alpha \int\limits_{[0,1)} f(x)\,dx &= \lim_{i\to\infty} \alpha_i \int\limits_{[0,1)} f(x)\,dx \overset{(8.1)}{=} \lim_{i\to\infty} \int\limits_{I_i} f(x)\,dx \overset{(8.2)}{=} \lim_{i\to\infty} g(p(I_i)) \\ &= \lim_{j\to\infty} g(p(I_j')) = g(\lim_{j\to\infty} p(I_j')) = g(\tilde{a}) = \int\limits_{\tilde{I}} f(x)\,dx.\end{aligned}$$

□

We now show that the number of intervals $2n - 1$ in the preceding theorem is tight.

Proposition 8.13. Let $n \geq 2$. There is some $\alpha \in [0,1]$ and some $f\colon [0,1) \to \mathbb{R}^{2n}$ such that for any $I \in \mathcal{I}_{2n-2}$

$$\int_I f(x)\,dx \neq \alpha \int_{[0,1)} f(x)\,dx.$$

Proof. Let $n \geq 2$. Define $p = \frac{1}{2n-1}$ and choose any $0 < \alpha < p$. For $k \in \{0,1,\ldots,2n-2\}$ let $P_k = [kp, kp + \frac{p}{2}) \in \mathcal{I}$. Now define the function $f(x) = (f_1(x), \ldots, f_{2n}(x))^{\mathrm{T}}$ where

$$f_1(x) = 1 \quad \text{and}$$
$$f_{k+2}(x) = \begin{cases} 1 & \text{if } x \in P_k \\ 0 & \text{otherwise} \end{cases}$$

for $k = 0, \ldots, 2n-2$. Let $I \in \mathcal{I}_r$ for some $r \in \mathbb{N}$ such that

$$\int_I f(x)\,dx = \alpha \int_{[0,1)} f(x)\,dx.$$

We will show that $r \geq 2n-1$. Observe that there is no set $J \subseteq I$, $J \in \mathcal{I}$ such that J intersects with two different P_{k_1}, P_{k_2} where $k_1, k_2 \in \{0,1,\ldots,2n-2\}$, since otherwise, $\int_I f_1(x)\,dx \geq p > \alpha = \alpha \int_{[0,1)} f_1(x)\,dx$. On the other hand, since for each $k \in \{0,1,\ldots,2n-2\}$ there must be at least one set $J \subseteq I$, $J \in \mathcal{I}$ that intersects with P_k, we must have $r \geq 2n-1$. □

8.3 Balancing Multiple Functions

When we invest even more intervals, we can partition $[0,1)$ into parts such that each part $I_i \subseteq [0,1)$ attains a pre-specified ratio of the value of the whole interval (Lemma 8.14). This result is then used to show that for a number of functions $f_1, \ldots, f_m\colon [0,1) \to \mathbb{R}^{2n}$ and a set of ratios $\alpha_1, \ldots, \alpha_m$ that sum up to 1, we can partition $[0,1)$ into m sets I_i (made up of a bounded number of intervals) such that for each i,

$$\int_{I_i} f_i(x)\,dx = \alpha_i \int_{[0,1)} f_i(x)\,dx.$$

Lemma 8.14. Let $n \geq 1$ and $f\colon [0,1) \to \mathbb{R}^{2n}$ be componentwise integrable. For every $m \geq 2$ and $\alpha \in [0,1]^m$ with $\|\alpha\|_1 = 1$, the set $[0,1)$ can be partitioned into sets $I_1, I_2, \ldots, I_m \in \mathcal{I}_{(m-1)(2n-1)}$ such that

$$\int_{I_i} f(x)\,dx = \alpha_i \int_{[0,1)} f(x)\,dx$$

for all i.

Proof. We inductively construct sets $I'_i \in \mathcal{I}_{2n-1}$, define $I_j := I'_j - (I'_1 \cup \cdots \cup I'_{j-1})$, and show that $I_j \in \mathcal{I}_{(m-1)(2n-1)}$.

Induction Basis ($i = 1$): We apply Theorem 8.12 with factor α_1 to f and obtain some $I_1 = I'_1 \in \mathcal{I}_{2n-1}$ such that

$$\int_{I'_1} f(x)\,dx = \alpha_1 \int_{[0,1)} f(x)\,dx.$$

Induction Step ($i \to i+1$ for $i+1 < m$): For $1 \le j \le i$ we have already defined the sets I'_j and $I_j = I'_j - (I'_1 \cup \cdots \cup I'_{j-1})$ such that

$$\int_{I_j} f(x)\,dx = \alpha_j \int_{[0,1)} f(x)\,dx.$$

Let $g\colon [0,1) \to \mathbb{R}^{2n}$ such that

$$g(x) = \begin{cases} f(x) & \text{if } x \notin I_1 \cup I_2 \cup \cdots \cup I_i \\ 0 & \text{otherwise.} \end{cases}$$

Note that the I_j are pairwise disjoint by definition and thus we get

$$\int_{[0,1)} g(x)\,dx = \int_{[0,1)-(I_1\cup\cdots\cup I_i)} f(x)\,dx = (1 - \sum_{j=1}^{i} \alpha_j) \int_{[0,1)} f(x)\,dx.$$

We assume that $\sum_{j=1}^{i} \alpha_j < 1$ (otherwise, $\alpha_{i+1} = 0$, hence $I'_{i+1} = \emptyset$) and apply Theorem 8.12 with factor $\frac{\alpha_{i+1}}{1-\sum_{j=1}^{i}\alpha_j}$ to g. We obtain some $I'_{i+1} \in \mathcal{I}_{2n-1}$ such that

$$\int_{I'_{i+1}} g(x)\,dx = \frac{\alpha_{i+1}}{1 - \sum_{j=1}^{i} \alpha_j} \int_{[0,1)} g(x)\,dx = \alpha_{i+1} \int_{[0,1)} f(x)\,dx.$$

Furthermore, by the definition of g, for $I_{i+1} := I'_{i+1} - (I'_1 \cup \cdots \cup I'_i)$ it holds that

$$\int_{I_{i+1}} g(x)\,dx = \int_{I'_{i+1}} g(x)\,dx = \alpha_{i+1} \int_{[0,1)} f(x)\,dx.$$

Last Step ($i = m$): Let $I_m := [0,1) - (I'_1 \cup \cdots \cup I'_{m-1})$. Observe that $I_1 \cup \cdots \cup I_{m-1} = I'_1 \cup \cdots \cup I'_{m-1}$ and

$$\int_{I_1\cup\cdots\cup I_{m-1}} f(x)\,dx = \sum_{j=1}^{m-1} \alpha_j \int_{[0,1)} f(x)\,dx = (1 - \alpha_m) \int_{[0,1)} f(x)\,dx.$$

This shows

$$\int_{I_m} f(x)\,dx = \alpha_m \int_{[0,1)} f(x)\,dx.$$

By construction, the sets I_i partition $[0,1)$. So it remains to argue that each of them can be expressed as a union of $(m-1)(2n-1)$ sets from $\mathcal{I}$. Obviously, if I is an arbitrary union of k sets from $\mathcal{I}$, the set $I - X$ for any $X \in \mathcal{I}$ can be expressed as a union of at most $k+1$ sets from $\mathcal{I}$.

If $i \leq m-1$, then $I_i = I'_i - (I'_1 \cup \cdots \cup I'_{i-1})$ is obtained from I'_i by removing at most $m-2$ unions of $2n-1$ sets from $\mathcal{I}$. So at most $(m-2)(2n-1)$ sets from $\mathcal{I}$ are removed from I'_i. Thus I_i can be expressed as a union of at most $(2n-1)+(m-2)(2n-1) = (m-1)(2n-1)$ sets from $\mathcal{I}$.

Finally, observe that since $I'_1 \cup \cdots \cup I'_{m-1} \in \mathcal{I}_{(m-1)(2n-1)}$, by Property 8.7, also $I_m = [0,1) - (I'_1 \cup \cdots \cup I'_{m-1}) \in \mathcal{I}_{(m-1)(2n-1)}$. □

Theorem 8.15. Let $n \geq 1$, $f_1, \ldots, f_m \colon [0,1) \to \mathbb{R}^{2n}$ componentwise integrable, and $\alpha \in [0,1]^m$ with $\|\alpha\|_1 = 1$. The set $[0,1)$ can be partitioned into sets $I_1, \ldots, I_m \in \mathcal{I}_{(m-1)(2mn-1)}$, such that for all $i \in \{1, \ldots, m\}$,

$$\int_{I_i} f_i(x)\,dx \;=\; \alpha_i \int_{[0,1)} f_i(x)\,dx.$$

Proof. Let $g \colon [0,1) \to \mathbb{R}^{2mn}$, $x \mapsto f_1(x) \circ f_2(x) \circ \cdots \circ f_m(x)$, where $\circ$ denotes the concatenation of vectors, i. e., $(x_1, \ldots, x_s)^{\mathrm{T}} \circ (y_1, \ldots, y_t)^{\mathrm{T}} = (x_1, \ldots, x_s, y_1, \ldots, y_t)^{\mathrm{T}}$. Clearly, g is componentwise integrable. By Lemma 8.14, there is a partition of $[0,1)$ into sets $I_1, I_2, \ldots, I_m \in \mathcal{I}_{(m-1)(2mn-1)}$ such that $\int_{I_i} g(x)\,dx = \alpha_i \int_{[0,1)} g(x)\,dx$ for all i. In particular this holds for the components of g corresponding to f_i. □

8.4 Application to Discrepancy Theory

We now want to transfer the exact analytical results from the previous sections to results about functions $f \colon \{1, \ldots, n\} \to \mathbb{R}^m$. We can consider f to be a function on $[0,1)$ that is constant on $[\frac{i}{n}, \frac{(i+1)}{n})$ for each $i = 0, \ldots, n-1$. Then most parts of the integrals degrade into simple sums. Since we cannot force the interval boundaries to be in $\{\frac{i}{n} \mid i \in \mathbb{N}, i < n\}$, we have to admit small errors when we really want to have sums everywhere. Fortunately, since we have rather few intervals, the number of interval boundaries that have to be moved to points of the form $\frac{i}{n}$ are also rather small.

More formally, we will show a result concerning weighted discrepancy. We will follow the notation used by Doerr [Doe05] to define several discrepancy notions. Remember that 1_n is the all-ones-vector of dimension n.

Definition 8.16. For a matrix $A \in \mathbb{R}^{m \times n}$, we define the *weighted discrepancy*, the *linear discrepancy* and the *hereditary linear discrepancy* as follows:

$$\begin{aligned}
\operatorname{wdisc}(A) &= \max_{\alpha \in [0,1]} \min_{x \in \{0,1\}^n} \|A(\alpha 1_n - x)\|_\infty \\
\operatorname{lindisc}(A) &= \max_{y \in [0,1]^n} \min_{x \in \{0,1\}^n} \|A(y - x)\|_\infty \\
\operatorname{herlindisc}(A) &= \max_{A_0 \leq A} \operatorname{lindisc}(A_0) = \max_{A_0 \leq A} \max_{y \in [0,1]^n} \min_{x \in \{0,1\}^n} \|A(y - x)\|_\infty,
\end{aligned}$$

where $A_0 \leq A$ means that A_0 is a submatrix of A, i. e. one obtains A_0 from A by deleting some rows and columns. Observe that it suffices to delete columns here. Further note that for all $A \in \mathbb{R}^{n \times m}$, we have $\operatorname{wdisc}(A) \leq \operatorname{lindisc}(A) \leq \operatorname{herlindisc}(A)$.

We did not require x to be a rounding of y in the definition of the linear discrepancy. Doerr [Doe05] remarks that the hereditary linear discrepancy is the correct notion for the minimal error introduced by rounding, as it holds that

$$\operatorname{herlindisc}(A) = \max_{y \in [0,1]^n} \min_{x \in \operatorname{rd}(y)} \|A(y-x)\|_\infty,$$

where $\operatorname{rd}(y) = \{x \in \{0,1\}^n \mid \forall i\colon |y_i - x_i| < 1\}$ is the set of roundings of y. The theorem by Beck and Fiala can now be formulated as follows (cf. Theorem 8.1):

Theorem 8.17 (Beck, Fiala [BF81])**.** For all $A \in \mathbb{R}^{m \times n}$ it holds that

$$\operatorname{herlindisc}(A) \leq \|A\|_1.$$

We already remarked that the weighted discrepancy of a matrix $A \in \mathbb{R}^{m \times n}$ can be bounded by the hereditary linear discrepancy. We use the results from the previous sections to show that this bound can always be achieved by a vector $x \in \{0,1\}^n$ that has a very special structure: The number of changes in the components of x is at most $2m$ and thus independent of n.

Note that the presence of this special structure has an important implication for rounding algorithms: If m is at most logarithmic in the input length and n at most polynomial, a vector $x \in \{0,1\}^n$ such that $\|A(\alpha \cdot 1_n - x)\| \leq \operatorname{herlindisc}(A)$ can always be found in polynomial time by simple exhaustive search. This also means that non-constructive proofs for upper bounds for the hereditary linear discrepancy are sufficient to improve such algorithms that use the weighted discrepancy.

Theorem 8.18. For all $A \in \mathbb{R}^{m \times n}$ and $\alpha \in [0,1]$ there is some $x \in \{0,1\}^n$ such that $x_k \neq x_{k+1 \pmod n}$ holds for at most $2m$ of the $k \in \{1, \dots, n\}$ and

$$\|A(\alpha \cdot 1_n - x)\|_\infty \leq \operatorname{herlindisc}(A).$$

Proof. Let $A = (a_{i,j})$ and define $f\colon [0,1) \to \mathbb{R}^m$ such that for $1 \leq j \leq n$ and $x \in [\frac{j-1}{n}, \frac{j}{n})$, $f(x) = (a_{1,j}, \dots, a_{m,j})^{\mathrm{T}}$. Note that f is componentwise integrable. We apply Theorem 8.12 and obtain a set $I \in \mathcal{I}_m$ such that

$$\int_I f(x)\,dx = \alpha \int_{[0,1)} f(x)\,dx.$$

Note that since m can be odd here, we have to add a zero component to f in that case and only obtain $I \in \mathcal{I}_m$ and not $I \in \mathcal{I}_{m-1}$.

We now define $p \in [0,1]^n$ as $p_j = n \cdot \lambda(I \cap [j-1, j))$ for $1 \leq j \leq n$ where λ is the Lebesgue measure, i. e. $\lambda(X) := \int_X 1\,dx$. We obtain

$$\begin{aligned} Ap &= \sum_{j=1}^{n} f(\tfrac{j-1}{n}) \cdot n \cdot \lambda(I \cap [j-1, j)) = n \int_I f(x)\,dx \\ &= n \cdot \alpha \int_{[0,1)} f(x)\,dx = \alpha \sum_{j=1}^{n} f(\tfrac{j-1}{n}) = A\,\alpha\,1_n. \end{aligned} \tag{8.3}$$

Next, we round the non-integral components of p. Observe that p_j is non-integral only if there is an endpoint of some interval of I in $[j-1, j)$. There are at most $2m$ endpoints of intervals, so p can have at most $2m$ non-integral components. Furthermore, if there is no endpoint of an interval in $[j-1, j+1)$, then $p_j = p_{j+1}$. This holds similarly for p_m and p_1. This means that p contains at most $2m$ pairs of adjacent components such that one of the components is non-integral or the two components differ (as the special case where the endpoint of an interval is some $\frac{j-1}{n}$). We will get back to this observation later.

We extract the columns from A that correspond to non-integral components of p. Let $A^{\text{fr}} \in \mathbb{R}^{m \times n}$ such that column j of A^{fr} is equal to column j of A if p_j is non-integral and all zeros otherwise. Let now $x \in \{0,1\}^n$ such that $\|A^{\text{fr}}(p-x)\|_\infty = \text{lindisc}(A^{\text{fr}}) \leq \text{herlindisc}(A)$. The inequality holds since A^{fr} corresponds to a submatrix of A. We can assume that $x_j = p_j$ if p_j is integral. It now holds that

$$A(\alpha 1_n - x) \overset{(8.3)}{=} A(p-x) = A^{\text{fr}}(p-x)$$

and thus we get

$$\|A(\alpha 1_n - x)\|_\infty \leq \text{herlindisc}(A).$$

It remains to show that $x_k \neq x_{k+1 \pmod n}$ holds for at most $2m$ of the $k \in \{1, \ldots, n\}$. We have already noted that p contains at most $2m$ pairs of adjacent components such that one of the components is non-integral or the two components differ. Since x differs from p only in non-integral components of p, we get that there are at most $2m$ pairs of adjacent components in x that are not equal (including x_m and x_1), which proves the assertion. □

We note that the above theorem can be extended to multi-color discrepancy by exchanging the application of Theorem 8.12 by Theorem 8.15. Since the number of intervals increases in Theorem 8.15, we also have to admit more color changes, but they are still independent of n.

Part II

Language Equations

Any (non-empty) regular language can be represented by an expression that combines singleton constants using the operations of union, concatenation and iteration. Of course, for each regular language there can be several such *regular expressions*, and thus, it is natural to ask whether two given regular expressions represent the same language (the so-called *equivalence problem*). Furthermore, one can ask the same question for regular expressions extended by the operations of intersection, complementation or squaring. Meyer and Stockmeyer [MS72, SM73] investigated these problems and showed that their complexity heavily depends on the allowed operations. They also studied *membership problems*, where an expression and a word is given as input and the question is whether the word belongs to the language represented by the expression. If the considered alphabet consists of only one letter, these expressions can also be interpreted as *expressions over sets of natural numbers* (sometimes also called *integer expressions*): Since a word over a single-letter alphabet is completely characterized by its length, languages correspond to sets of natural numbers and the concatenation corresponds to addition. For natural numbers, multiplication can of course also be considered as an operation for these expressions. Some problems concerning expressions over sets of natural numbers were also studied by Meyer and Stockmeyer [SM73].

When we allow expressions to reference subexpressions of themselves, we arrive at a more succinct representation of the same languages. These "expressions" can be modeled by acyclic graphs (circuits) where each node represents one application of an operator to its predecessors and input gates represent singletons. These circuits were mostly studied over sets of natural numbers by Wagner [Wag84], Yang [Yan00] and McKenzie and Wagner [MW07]. In chapter 10, we investigate the complexity of equivalence problems for circuits over sets of natural numbers and show that they are complete for various complexity classes ranging from NL over $\mathrm{C_{=}L}$, P and Π_2^{P} up to PSPACE. We obtain

similar results for *satisfiability problems*, where the question is whether some assignment to the constants generates a specific number in the set represented by the circuit.

The circuit framework can be further extended by dropping the condition that the graph must be acyclic, which can be seen as follows. For each gate i in the circuit, we introduce one variable A_i and one expression $\varphi_i = A_{i_1} \otimes A_{i_2}$ representing the operation applied on the predecessor gates (we assume a binary operation for simplicity) or one expression $\varphi_i = \{a\}$ for input gates with assignment $a \in \Sigma$. These expressions can then be assembled into a system of (language) equations:

$$\begin{aligned} A_1 &= \varphi_1 \\ A_2 &= \varphi_2 \\ &\vdots \\ A_r &= \varphi_r \end{aligned}$$

Ginsburg and Rice [GR62] showed that if the allowed operations are union and concatenation, one obtains exactly the context-free languages as (least) solutions. By adding intersection and complementation to the set of operations, one arrives at *conjunctive* and *Boolean languages* defined by Okhotin [Okh01, Okh04] (the semantics are a bit delicate when complementation is allowed, we refer to section 9.1 for details). Although the classes of conjunctive and Boolean languages are strictly larger than the context-free languages, they share some convenient properties with the context-free languages, among these the existence of efficient parsing algorithms (i. e. algorithms solving the membership problems).

In section 9.2 we show that systems of language equations defining conjunctive languages do not need the full power of the union: In order to represent them, it suffices to allow arbitrary intersection and concatenation, and union only with singletons. In contrast, when the same restriction on the union is imposed on systems of equations as studied by Ginsburg and Rice (i. e. without intersection), one arrives at a proper subclass of the context-free languages. This class was studied by Greibach, Shi and Simonson [GSS92] under the name of *single tree grammars*.

Quite surprisingly, it was shown by Jeż [Jeż08] that even over a single-letter alphabet, the conjunctive languages are a strict superclass of the context-free languages (which coincide with the regular languages in this case). This initiated the study of the expressive power of such languages and also the study of the membership problem for systems of language equations over a single-letter alphabet (also including complementation), or, by the correspondence mentioned above, systems of equations over sets of natural numbers [JO11a, JO08]. It was shown that this problem is complete for E.

We investigate this problem in more detail in section 9.3 and give a sharper upper bound. More specifically, we present a parsing algorithm for Boolean languages over a single-letter alphabet. Similar to Valiant's reduction to matrix multiplication for parsing context-free languages, we give a sequence of reductions to arithmetic problems. This algorithm uses Boolean convolution to show that on input of a word a^n and a system of language equations S, the question whether the system generates a language including a^n can be solved in time $O(|S| \cdot M(n \log n) \cdot \log n)$. Here $M(n)$ denotes the bit-complexity of multiplying two integers of length n. The currently best known value for $M(n)$ [Für09] yields the complexity $|S| \cdot n \log^3 n \cdot 2^{O(\log^* n)} \leq |S| \cdot n \log^4 n$. Note that for the membership

problem considered by Jeż and Okhotin, the input a^n is encoded as $\mathrm{bin}(n)$ and thus we show that their problem lies in $\mathrm{DTIME}(2^n\, n^4)$.

Chapter 9

Conjunctive and Boolean Grammars

Context-free grammars are widely used for defining formal languages, or in other words, syntactic requirements as they for instance occur in programming or markup languages. In addition to the availability of efficient parsing algorithms for context-free languages, they are understandable by non-specialists: The set of rules $S \to aSb$, $S \to \varepsilon$ models the inductive requirement that a word w has property S if it is the empty word or can be written as avb, where the word v also has property S.

Multiple rules for one nonterminal are of course connected disjunctively, which can be seen by the word "or" in the informal description. As the conjunction of syntactical conditions is not expressible in context-free grammars, this model can be extended by allowing an explicit conjunction in the formalism of rules. The resulting extension, introduced by Okhotin [Okh01], is known as *conjunctive grammar*. It maintains the principle of defining a language inductively and still allows efficient parsing algorithms [Okh07, Okh08]. At the same time, using conjunction in addition to disjunction considerably increases the expressive power of the model. Besides being able to represent many standard examples of non-context-free languages, such as $\{a^n b^n c^n \mid n \in \mathbb{N}\}$, and even languages outside of the intersection-closure of the context-free languages, such as $\{wcw \mid w \in \{a,b\}^*\}$ [Okh01, Wot73], conjunctive grammars are notable for their non-trivial expressive power over a single-letter alphabet, studied by Jeż [Jeż08] and by Jeż and Okhotin [JO08, JO10, JO11a].

Similar to the extension by conjunction, it is worthwhile to consider grammars that include an explicit negation. These so-called *Boolean grammars* were also defined and studied by Okhotin [Okh04]. Regardless of their expressive power, both conjunctive and Boolean grammars preserve the efficient parsing techniques of the context-free languages. In particular, the membership of a word of length n in the language generated by a grammar G can be tested in time $\Theta(|G| \cdot n^3)$ by a straightforward adaptation of the Cocke–Kasami–Younger algorithm [Okh01, Okh04], and a more careful examination showed that Valiant's [Val75] reduction of context-free recognition to *Boolean matrix multiplication* is still applicable to Boolean grammars [Okh10], leading to a parsing algorithm working in time $O(|G| \cdot \mathrm{BMM}(n) \log n)$, where $\mathrm{BMM}(n)$ is the number of bit operations needed to multiply two $n \times n$ Boolean matrices [Okh10]. Using the best known upper bound on matrix multiplication [CW90] yields $O(|G| \cdot n^{2.376})$ time complexity of parsing.

We begin this chapter with an introduction on conjunctive and Boolean grammars in section 9.1 that contains all needed definitions and also explains the interesting case of

single-letter alphabets. Then we show two main results in the subsequent sections.

9.1 Definitions and Normal Forms

We define conjunctive and Boolean languages as an extension of context-free languages through their grammatical characterizations. The semantics for conjunctive languages can be defined via derivations and this will turn out useful for proofs, as we can use induction over the number of derivation steps. Though there is some intuition for derivations with negation, we will define the semantics for Boolean grammars only via solutions to systems of language equations, as mentioned in the introduction to this part. For an overview of conjunctive and Boolean grammars and languages we refer to the recent survey by Okhotin [Okh11].

Definition 9.1 (Okhotin [Okh01],[Okh04])**.** A *Boolean grammar* is a quadruple $G = (\Sigma, N, S, R)$, in which Σ and N are disjoint finite nonempty sets of terminal and nonterminal symbols, respectively, $S \in N$ is the start symbol and R is a finite set of rules, each of the form

$$A \to \alpha_1 \& \dots \& \alpha_n \& \neg\beta_1 \& \dots \& \neg\beta_m \qquad \text{where } A \in N,\ n, m \in \mathbb{N},\ n+m \geq 1 \text{ and } \alpha_1, \dots, \alpha_n, \beta_1, \dots, \beta_m \in (\Sigma \cup N)^*. \tag{9.1}$$

Here, "$\to$", "&" and "$\neg$" are special symbols not in $\Sigma \cup N$. If the symbol $\neg$ does not occur in any rule, G is called a *conjunctive grammar*. In such a rule given above, α_i and $\neg\beta_i$ are called *conjuncts*. Multiple rules $A \to \varphi$, $A \to \psi$ can be denoted as $A \to \varphi \mid \psi$. A rule (9.1) is called *terminating* if it is of the form $A \to w$ with $w \in \Sigma^*$, and *non-terminating* otherwise.

Informally, a rule (9.1) states that if a word is generated by each α_i and none of the β_i, then it is generated by A. Because of rules of the type $A \to \neg A$ and other reasons, the formalization of this concept is a bit more complicated for Boolean grammars. For conjunctive grammars, the semantics can be defined similar to context-free grammars by stepwise derivations or, more generally, term rewriting, which generalizes Chomsky's string rewriting [BKV03].

Definition 9.2 ([Okh01])**.** Given a conjunctive grammar $G = (\Sigma, N, S, R)$, consider terms over concatenation (not explicitly denoted) and conjunction (denoted by "&") with symbols from $\Sigma \cup N$ as atomic terms. Assume that the symbols "(" and ")" used for grouping are not in $\Sigma \cup N$. The relation $\overset{G}{\Longrightarrow}$ of derivability in one step on the set of terms is defined as follows:

- Some rule $A \to \alpha_1 \& \dots \& \alpha_n \in R$ can be applied to a term t_1 that contains A: If the term t_2 is obtained from t_1 by replacing one occurrence of A in t_1 by $(\alpha_1 \& \dots \& \alpha_n)$ then we can write $t_1 \overset{G}{\Longrightarrow} t_2$.
- A conjunction of several identical terminal words can be rewritten by one such word: If the term t_1 contains $(w \& \dots \& w)$ for some $w \in \Sigma^*$ and t_2 is obtained from t_1 by replacing $(w \& \dots \& w)$ by the single word w then we can write $t_1 \overset{G}{\Longrightarrow} t_2$.

If the grammar G is clear from the context, we also write $\Longrightarrow$ for $\overset{G}{\Longrightarrow}$. As usual, the relations of *derivability (in zero or more steps) and in exactly ℓ steps*, are denoted by $\overset{G}{\Longrightarrow}^*$ and $\overset{G}{\Longrightarrow}^\ell$, respectively. The *language generated by a term φ* is $L_G(\varphi) = \{w \in \Sigma^* \mid \varphi \overset{G}{\Longrightarrow}^* w\}$. The *language generated by the grammar* is $L(G) = L_G(S) = \{w \in \Sigma^* \mid S \overset{G}{\Longrightarrow}^* w\}$. A language L that is generated by some conjunctive grammar is called a *conjunctive language*.

The semantics of Boolean grammars are usually defined using language equations. This definition generalizes the well-known characterization of the context-free grammars by equations, due to Ginsburg and Rice [GR62] and for conjunctive grammars, it is equivalent to the one given above.

Definition 9.3 ([Okh04])**.** For every Boolean grammar $G = (\Sigma, N, S, R)$ the *associated system of language equations* contains the equation

$$A = \bigcup_{\substack{A \to \alpha_1 \& \ldots \& \alpha_n \& \\ \neg\beta_1 \& \ldots \& \neg\beta_m \in R}} \left[\bigcap_{i=1}^{n} \alpha_i \cap \bigcap_{j=1}^{m} \overline{\beta_j} \right] \tag{9.2}$$

for each $A \in N$ where each symbol in N is interpreted as a variable, each symbol $a \in \Sigma$ is interpreted as the constant language $\{a\}$ and each empty word as the constant language $\{\varepsilon\}$. A *solution* to such a system is a vector of languages $(L_A)_{A \in N}$, $L_A \subseteq \Sigma^*$, such that the substitution of L_A for A, for each $A \in N$, turns each equation (9.2) into an equality, where the complement is interpreted relative to Σ^*. A solution $(L_A)_{A \in N}$ is called *strongly unique* if it is the only solution and furthermore for each $n \in \mathbb{N}$, the vector $(L_A \cap \Sigma^{\leq n})_{A \in N}$ is a solution to the system that is obtained by intersecting each right-hand side with $\Sigma^{\leq n}$. If $(L_A)_{A \in N}$ is a strongly unique solution, then we can speak of the *language generated by* $A \in N$ as $L_G(A) = L_A$ and the *language generated by* G as $L(G) = L_G(S)$. A language L that is generated by some Boolean grammar is called a *Boolean language*.

Note that by the definition, only the Boolean grammars that have strongly unique solutions generate languages. The idea behind the concept of strongly unique solutions is that we want to retain the inductive character of grammars: The question whether a word w is generated by a grammar does not depend on words longer than w. Observe that for a general Boolean grammar G, the associated system of language equations does not need to have a unique solution (a simple example is the grammar that contains only the rule $S \to S$) and sometimes does not even have any solution (for instance the grammar with the rule $S \to \neg S$). For a detailed discussion of both of these aspects, we refer to the article by Okhotin [Okh04]. Note that there is also another way to define languages generated by Boolean grammars that uses three-valued logics [KNR09].

9.1.1 Examples of Conjunctive and Boolean Grammars

Let us now give some examples of conjunctive and Boolean grammars. Every language representable as an intersection of finitely many context-free languages such as $\{a^n b^n c^n \mid n \in \mathbb{N}\}$, can be straightforwardly specified using conjunction for the start symbol, as demonstrated in the following grammar.

Example 9.4 (Okhotin [Okh01])**.** The conjunctive grammar defined by the following rules generates the language $\{a^n b^n c^n \mid n \in \mathbb{N}\}$:

$$\begin{array}{lcl} S & \to & AB\&DC \\ A & \to & aA \mid \varepsilon \\ B & \to & bBc \mid \varepsilon \\ C & \to & cC \mid \varepsilon \\ D & \to & aDb \mid \varepsilon \end{array}$$

Here $L(AB) = \{a^i b^m c^m \mid i, m \in \mathbb{N}\}$ and $L(DC) = \{a^k b^k c^j \mid k, j \in \mathbb{N}\}$, while the intersection of these two languages represented in the rule for S is exactly $\{a^n b^n c^n \mid n \in \mathbb{N}\}$.

As the conjunction in the above example is not applied recursively, the grammar can only generate a language in the intersection closure of the context-free languages. It is more interesting to construct a grammar for languages outside of this intersection closure, such as the following one.

Example 9.5 (Okhotin [Okh01])**.** The conjunctive grammar defined by the rules

$$\begin{array}{lcl} S & \to & C\&D \\ C & \to & aCa \mid aCb \mid bCa \mid bCb \mid c \\ D & \to & aA\&aD \mid bB\&bD \mid cE \\ A & \to & aAa \mid aAb \mid bAa \mid bAb \mid cEa \\ B & \to & aBa \mid aBb \mid bBa \mid bBb \mid cEb \\ E & \to & aE \mid bE \mid \varepsilon \end{array}$$

generates the language $\{wcw \mid w \in \{a, b\}^*\}$. In particular, $L(D) = \{uczu \mid u, z \in \{a, b\}^*\}$. The rules for D match a single symbol in the left part to the corresponding symbol in the right part using A or B, and the recursive reference to aD or bD makes the remaining symbols be compared in the same way. The intersection with the language $\{ucv \mid u, v \in \{a, b\}^*, |u| = |v|\}$ generated by C completes the grammar.

Note that $\{wcw \mid w \in \{a, b\}^*\}$ cannot be written as a (finite) intersection of context-free languages [Wot73].

It is not known whether the language $\{ww \mid w \in \{a, b\}^*\}$ can be generated by a conjunctive grammar, but a Boolean grammar for this language is known [Okh04, Okh11].

Example 9.6 (Okhotin [Okh04])**.** The Boolean grammar defined by the rules

$$\begin{array}{lcl} S & \to & \neg AB\&\neg BA\&C \\ A & \to & XAX \mid a \\ B & \to & XBX \mid b \\ C & \to & XXC \mid \varepsilon \\ X & \to & a \mid b \end{array}$$

generates the language $\{ww \mid w \in \{a, b\}^*\}$. Note that $L(AB) = \{uavxby \mid u, v, x, y \in \{a, b\}^*, |u| = |v|, |x| = |y|\} = \{uavxby \mid u, v, x, y \in \{a, b\}^*, |u| = |x|, |v| = |y|\}$.

It should be noted that the greatest open problem for conjunctive and Boolean grammars is the search for a (nontrivial) technique to show that some language can not be generated by such a grammar. In particular, it is unknown if there is a Boolean language that is not conjunctive. For more details, we refer to the recent survey by Okhotin [Okh11].

9.1.2 Single-Letter Alphabets

It is well-known that over a single-letter alphabet the regular and the context-free languages coincide [GR62]. Thus, the expressive power of context-free languages over a single-letter alphabet is very limited. Surprisingly, this is not true for conjunctive grammars, as will be seen in the following example.

Example 9.7 (Jeż [Jeż08])**.** The following conjunctive grammar with the start symbol A_1 generates the (non-regular) language $\{a^{4^n} \mid n \in \mathbb{N}\}$:

$$\begin{aligned} A_1 &\to A_1A_3\&A_2A_2 \mid a \\ A_2 &\to A_1A_1\&A_2A_6 \mid aa \\ A_3 &\to A_1A_2\&A_6A_6 \mid aaa \\ A_6 &\to A_1A_2\&A_3A_3 \end{aligned}$$

In particular, each nonterminal A_i generates the language $\{a^{i\cdot 4^n} \mid n \in \mathbb{N}\}$.

To get a feeling how this grammar works, one can consider the set of lengths of the words generated by a nonterminal in base 4 notation. Ignoring the problem of leading zeros for the moment, we can consider a language $L \subseteq \{a\}^*$ equivalent to the set of the word lengths in base 4 notation, i. e. $L \mathrel{\hat{=}} \{w \in \{0,1,2,3\}^* \mid a^n \in L \text{ where } w \text{ is the base 4 notation of } n \in \mathbb{N}\}$. Then $L(A_1) \mathrel{\hat{=}} 10^*$, $L(A_2) \mathrel{\hat{=}} 20^*$, $L(A_3) \mathrel{\hat{=}} 30^*$ and $L(A_6) \mathrel{\hat{=}} 120^*$ (note that 6 is 12 in base 4). This is useful since the concatenation of languages over a single-letter alphabet corresponds to the (element-wise) addition of the sets of word lengths (we already explained this correspondence in the introduction to this part while clarifying the connections between circuits and equations over languages and over sets of natural numbers). We now show that the stated languages are a solution to the first equation. Substituting the solutions, we obtain $L(A_1A_3) = 10^*30^* \cup 30^*10^* \cup 100^*$, where the last part is the intended set and the rest will be removed. Similarly, we get $L(A_2A_2) = 20^*20^* \cup 100^*$. The intersection of languages over $\{a\}^*$ directly corresponds to the intersection of the base 4 notation of the word lengths and thus we get $L(A_1A_3\&A_2A_2) = 100^*$, which is exactly the stated language generated by A_1 when $\{a\}$ is additionally included in the set.

This technique of manipulating the base k notation has subsequently been extended by Jeż and Okhotin. In particular, they showed that a language $L \subseteq \{a\}^*$ is conjunctive if there is some $k \geq 2$ such that $\{w \in \{0,1,\ldots,k-1\} \mid a^n \in L \text{ where } w \text{ is the base } k \text{ notation of } n \in \mathbb{N}\}$ is linear conjunctive [JO10]. Linear conjunctive languages are a certain subclass of the conjunctive languages that include the regular languages [Okh11]. Furthermore, for any recursive function there is a strictly greater function $f\colon \mathbb{N} \to \mathbb{N}$ such that the language $\{a^{f(n)} \mid n \in \mathbb{N}\}$ is conjunctive [JO10]. That means that the conjunctive languages over single-letter alphabets can grow as fast as any recursive function. Note that since conjunctive languages are decidable, their growth functions are all recursive. This result has to be compared to the fact that context-free languages over a single-letter alphabet can only grow linearly.

Finally, several complexity aspects concerning conjunctive grammars over single-letter alphabets have been studied. Among other results, it was established that testing whether two given conjunctive grammars over a single-letter alphabet generate the same language (i. e. the equivalence problem for the respective systems of equations over sets of natural

numbers) is coRE-complete [JO10]. Notably, this problem stays coRE-complete even when only a single nonterminal is allowed [JO11b]. Furthermore, the membership problem for systems of equations over sets of natural numbers with union, intersection and addition is E-complete [JO11a]. In section 9.3 we will give a sharper explicit upper bound on this membership problem (though note that we use a different input encoding) and we will study the complexity of equivalence problems for systems of equations over sets of natural numbers without cyclic dependencies between the variables in section 10.3.

9.1.3 Normal Forms

The Chomsky normal form for context-free grammars can be generalized to conjunctive and Boolean grammars as follows.

Definition 9.8 (Binary normal form [Okh01, Okh04])**.** A Boolean or conjunctive grammar $G = (\Sigma, N, S, R)$ is in *binary normal form* if every rule in R has one of the following three forms:

$A \to B_1C_1\& \dots \& B_nC_n\&$ $\quad \neg D_1E_1\& \dots \& \neg D_mE_m\& \neg\varepsilon$	for $n \geq 1, m \geq 0,\ B_i, C_i, D_i, E_i \in N$, where "$\&\neg\varepsilon$" can be omitted if $m = 0$
$A \to a$	for some $a \in \Sigma$
$S \to \varepsilon$	only if S does not appear in right-hand sides of rules

Every Boolean grammar that generates a language (i. e. the associated system of language equations has a strongly unique solution) and every conjunctive grammar can be effectively transformed to a grammar in binary normal form generating the same language [Okh01, Okh04] (while conjunctive grammars are transformed to conjunctive grammars). The technique to achieve this is essentially the same as for the context-free grammars. Observe that every grammar in binary normal form generates a language. In particular, for both conjunctive and Boolean grammars G, the set $L(G) \cap \Sigma^{\leq n+1}$ is obtained from $L(G) \cap \Sigma^{\leq n}$ by one application of the rules.

In the following, we will assume that Boolean grammars that are used as inputs for algorithms are given in binary normal form. This is reasonable, since it is undecidable if a given system of language equations has a strongly unique solution [Okh04].

The binary normal form can be used to obtain a simple generalization of the Cocke–Kasami–Younger parsing algorithm to conjunctive and Boolean grammars, which still works in time $O(|G| \cdot n^3)$ [Okh01, Okh04]. Note that the more efficient algorithm for parsing context-free grammars due to Valiant [Val75] can also be generalized to Boolean grammars and yields an $O(|G| \cdot \mathrm{BMM}(n) \log n)$-time algorithm, where $\mathrm{BMM}(n)$ is the complexity of multiplying two $n \times n$ Boolean matrices.

For context-free grammars, there is another important normal form: the *Greibach normal form* [Gre65], in which every rule is either $A \to a\alpha$ with $a \in \Sigma$ and $\alpha \in (\Sigma \cup N)^*$, or $A \to \varepsilon$. This definition naturally carries on to conjunctive grammars. Furthermore, one can also consider a more restrictive deterministic variant.

Definition 9.9. A conjunctive grammar $G = (\Sigma, N, S, R)$ is in *Greibach normal form* if every rule in R is of the form

$$A \to a\alpha_1 \& \ldots \& a\alpha_n \quad (n \geq 1,\ a \in \Sigma,\ \alpha_i \in N^*) \quad \text{or}$$
$$A \to \varepsilon.$$

The grammar G is in *deterministic Greibach normal form*, if for every $A \in N$ and $a \in \Sigma$, there is at most one rule of the form $A \to a\alpha_1 \& \ldots \& a\alpha_n$.

It is not known whether every conjunctive grammar can be transformed to this form [Okh11]. The only progress in this direction is the result that every conjunctive grammar over a single-letter alphabet can be transformed to Greibach normal form (cf. Corollary 9.16).

9.2 Restricted Conjunctive Grammars

This section continues the investigation of the power of Boolean operations in formal grammars with a subclass of conjunctive grammars in which the disjunction can be used only in the form of disjunction with a terminal word. In other words, each nonterminal A may have only one rule referring to nonterminals, while the rest of its rules must be of the form $A \to w$, where w is a terminal word. The same restriction on the context-free grammars has been studied by Greibach et al. [GSS92] under the name of *single tree grammars* because all parse trees in such a grammar can be obtained from a single infinite parse tree by stopping derivations at different subsets of its nodes.

These grammars have quite a limited expressive power; in particular, they cannot generate the language of all palindromes and not even some regular languages but at the same time, some of their basic decision problems, such as intersection emptiness and ambiguity, are undecidable [GSS92].

Similarly to single tree grammars, one can expect conjunctive grammars restricted to use disjunction only with terminal words to be much weaker than conjunctive grammars of the general form. However, the results of this section contrast this intuition, and it is shown that in fact every conjunctive grammar can be effectively transformed to an equivalent grammar with restricted disjunction. Unrestricted disjunction is thus redundant in conjunctive grammars, as opposed to context-free grammars. The form with restricted disjunction may be regarded as a normal form for conjunctive grammars.

After formally defining the restriction we want to consider, we will show the basic idea for the general transformation by giving a restricted grammar for all palindromes of odd length. This idea is applicable to any grammar in which no nonterminal symbols generate any words of even length. As an intermediate step, we show that any conjunctive grammar can be transformed into another normal form, the *odd normal form*, in which every nonterminal other than the start symbol generates only words of odd length. By combining both constructions, the main result of this section is obtained.

Finally, the question of eliminating ε-rules in conjunctive grammars with restricted disjunction is addressed. Though it is not determined whether this is always possible, a construction of restricted conjunctive grammars without ε-rules for a subfamily of conjunctive languages including all regular languages is given. Furthermore, it is shown

how the two important languages $\{wcw \mid w \in \{a,b\}^*\}$ and the (nonempty) palindromes can be generated by such grammars.

Definition 9.10. A *restricted conjunctive grammar* $G = (\Sigma, N, S, R)$ is a conjunctive grammar in which the set of rules for every nonterminal A is of the form

$$A \to \alpha_1 \& \ldots \& \alpha_n \mid w_1 \mid \ldots \mid w_m \qquad \text{for some } n \geq 1, m \geq 0, \alpha_i \in (\Sigma \cup N)^*, w_j \in \Sigma^*.$$

In other words, every nonterminal may have at most one non-terminating rule, that is, a rule *not* of the form $A \to w$, with $w \in \Sigma^*$. For instance, the grammar in Example 9.4 is restricted conjunctive, while the grammar in Example 9.5 is not. Since multiple rules for one nonterminal can be regarded as disjunction, this definition effectively requires every disjunction to have a terminal word (that is, a singleton constant language) as one of its (two) arguments.

The language of all palindromes of odd length over $\{a,b\}$ is generated by the context-free grammar defined by the rules

$$S \to aSa \mid bSb \mid a \mid b.$$

Because of the disjunction $aSa \mid bSb$, this grammar is not a single tree grammar. Indeed, this language cannot be generated by a single tree grammar. One can further prove that it is not representable as a finite union of languages defined by single tree grammars [Rei07] (the existing proof applies to the language of all palindromes, but it can be easily modified for the stated language). On the other hand, the next example shows that restricted conjunctive grammars can generate this language.

Example 9.11. The following restricted conjunctive grammar generates the set of palindromes of odd length over $\{a,b\}$:

$$\begin{aligned} S &\to AB\&O \mid a \mid b \\ A &\to aSa \mid \varepsilon \\ B &\to bSb \mid \varepsilon \\ O &\to OOO \mid a \mid b \end{aligned}$$

Here the nonterminal O generates the language $\textsc{Odd} := \{w \in \{a,b\}^* \mid |w| \text{ is odd}\}$, and hence S may generate only words of odd length. Then the rule $S \to AB\&O$ generates

$$(aSa \cup \{\varepsilon\})(bSb \cup \{\varepsilon\}) \cap \textsc{Odd} = (aSabSb \cup aSa \cup bSb \cup \{\varepsilon\}) \cap \textsc{Odd} = aSa \cup bSb,$$

that is, it is equivalent to two rules $S \to aSa$ and $S \to bSb$. Thus the set of odd-length palindromes is generated inductively, starting from a and b.

This representation of the union of two languages actually works in the general context, as long as both languages consist of words of odd length. As in the above example, it is sufficient to add the empty word to both languages, concatenate them and then filter out the words of even length:

$$(K \cup \{\varepsilon\})(L \cup \{\varepsilon\}) \cap \textsc{Odd} = K \cup L \quad (\text{for all } K, L \subseteq \textsc{Odd}) \tag{9.3}$$

This identity gives a way to simulate every conjunctive grammar, in which every nonterminal generates a subset of Odd, and the next task is to ensure that the latter condition holds.

9.2.1 Odd Normal Form

The new normal form for conjunctive grammars proposed in this section has the following main property: Every nonterminal (possibly except the start symbol) may only generate words of odd length. As the parity of the length of words is going to play an important role in all constructions below, let us introduce the notation $\textsc{Even} := (\Sigma^2)^*$ and $\textsc{Odd} := \Sigma(\Sigma^2)^*$ (where Σ is the implicitly assumed alphabet) for the sets of all words of even and odd length, respectively.

Definition 9.12 (Odd normal form)**.** A conjunctive grammar $G = (\Sigma, N, S, R)$ is said to be in *odd normal form* if all rules in R are of the form

$$\begin{aligned} &A \to a && \text{with } A \in N,\ a \in \Sigma, \qquad \text{or} \\ &A \to B_1 a_1 C_1 \,\&\, \ldots \,\&\, B_n a_n C_n && \text{with } n \geq 1,\ A, B_i, C_i \in N,\ a_i \in \Sigma. \end{aligned}$$

If S does not occur in the right-hand sides of the rules, then the following two types of rules, called *even rules*, are also allowed:

$$\begin{aligned} &S \to aA && \text{with } a \in \Sigma,\ A \in N \\ &S \to \varepsilon \end{aligned}$$

Note that if there are no even rules in a grammar in odd normal form, then it generates a subset of $\textsc{Odd}$. Thus even rules are needed for some languages, but regardless of whether they are used, the main part of the grammar operates on words of odd length only. The main step towards the transformation to the odd normal form is taking an arbitrary conjunctive grammar and representing its operation on arbitrary words using only words of odd length. The resulting grammar does not necessarily generate exactly the same language, but it preserves all the information in the grammar.

Lemma 9.13. For every conjunctive grammar $G = (\Sigma, N, S, R)$ there exists and can be effectively constructed a conjunctive grammar $G_1 = (\Sigma, N', S', R_1)$ in odd normal form without even rules, in which the set of nonterminals is $N' = (\Sigma \cup \{\varepsilon\}) \times N \times (\Sigma \cup \{\varepsilon\})$ where each nonterminal $(x, A, y) \in N'$ is denoted by ${}_xA_y$ and generates the language

$$L_{G_1}({}_xA_y) = x^{-1} L_G(A) y^{-1} \cap \textsc{Odd}.$$

The start symbol is $S' = {}_\varepsilon S_\varepsilon$, and hence $L(G_1) = L(G) \cap \textsc{Odd}$.

Proof. It can be assumed that G is in binary normal form, which can be obtained effectively [Okh01]. Furthermore, we may assume that G does not contain the rule $S \to \varepsilon$, since the languages $x^{-1} L_G(S) y^{-1} \cap \textsc{Odd}$ consist of words of length at least one, and hence the membership of ε in $L(G)$ does not affect them.

The transformation of G to G_1 is done in two steps. The main step yields a grammar G' that is *almost* in odd normal form, with occasional so-called *unit conjuncts*, that is, rules of the form $A \to \ldots \& B \& \ldots$. After that, G_1 is obtained from G' by a simple known procedure.

The intermediate grammar is defined as $G' = (\Sigma, N', S', R')$, with the set of nonterminals and the start symbol as in the statement of the lemma. For every rule

$$A \to B^{(1)} C^{(1)} \& \ldots \& B^{(n)} C^{(n)} \in R,$$

each nonterminal ${}_xA_y$ with $x, y \in \Sigma \cup \{\varepsilon\}$ in the new grammar G' has all possible rules of the form

$$ {}_xA_y \to {}_x\alpha_y^{(1)} \& \ldots \& {}_x\alpha_y^{(n)} $$

that satisfy the following condition for each i:

$$ {}_x\alpha_y^{(i)} \in \{ {}_xB_a^{(i)} \cdot a \cdot {}_\varepsilon C_y^{(i)} \mid a \in \Sigma \} \cup \tag{9.4a} $$

$$ \{ {}_xB_\varepsilon^{(i)} \cdot a \cdot {}_a C_y^{(i)} \mid a \in \Sigma \} \cup \tag{9.4b} $$

$$ \{ {}_xB_\varepsilon^{(i)} \mid y \in L_G(C^{(i)}) \} \cup \tag{9.4c} $$

$$ \{ {}_\varepsilon C_y^{(i)} \mid x \in L_G(B^{(i)}) \}. \tag{9.4d} $$

Additionally, for every ${}_xA_y \in N'$ and $a \in \Sigma$ with $xay \in L_G(A)$, the new grammar contains the rule

$$ {}_xA_y \to a. \tag{9.5} $$

Since every conjunct in G' is of length 1 or 3, all words generated by nonterminals in N' must be of odd length, and in particular no nonterminal in N' generates the empty word.

Claim 9.14. For each ${}_xA_y \in N'$ and for every $w \in \Sigma^*$,

$$ w \in L_{G'}({}_xA_y) \qquad \text{if and only if} \qquad xwy \in L_G(A) \text{ and } w \in \textsc{Odd}. $$

Proof of the claim. The proof in each direction is by induction on the length of xwy.

"$\Rightarrow$": Let $w \in L_{G'}({}_xA_y)$; it has to be proved that $xwy \in L_G(A)$.

Induction basis $|xwy| = 1$: Assume $w \in L_{G'}({}_xA_y)$ for $|xwy| = 1$. Since no nonterminals in G' generate the empty word, $w \neq \varepsilon$ and thus $x = y = \varepsilon$. If w is generated directly by a rule of type (9.5), then $w = a$ and $xwy \in L_G(A)$ by the construction. At the same time, w cannot be generated by any rule ${}_xA_y \to {}_x\alpha_y^{(1)} \& \ldots \& {}_x\alpha_y^{(n)}$: indeed, the conjuncts ${}_x\alpha_y^{(i)}$ cannot be of type (9.4a) or (9.4b), since these only generate longer words, and the types (9.4c) and (9.4d) are also not applicable, because $x = y = \varepsilon$ and no nonterminal in G generates the empty word. This means that w can only be generated by a rule of type (9.5), and the induction basis is complete.

Induction step: Let $\ell \geq 1$ and assume that the assertion holds for all A, x, w and y with $|xwy| \leq \ell$. Let $w \in L_{G'}({}_xA_y)$ for some ${}_xA_y \in N'$ and $|xwy| = \ell + 1$. If w is generated directly by a rule (9.5), then $xwy \in L_G(A)$ by the construction of this rule. Assume that w is generated by a rule ${}_xA_y \to {}_x\alpha_y^{(1)} \& \ldots \& {}_x\alpha_y^{(n)}$, that is, $w \in L_{G'}({}_x\alpha_y^{(i)})$ for all $i = 1, \ldots, n$. Fix an ith conjunct ${}_x\alpha_y^{(i)}$ and consider its form.

Assume this conjunct is of the form (9.4a), that is, ${}_x\alpha_y^{(i)} = {}_xB_a^{(i)} \cdot a \cdot {}_\varepsilon C_y^{(i)}$ for some $a \in \Sigma$. Then w can be factorized into $w = uav$, where $u \in L_{G'}({}_xB_a^{(i)})$ and $v \in L_{G'}({}_\varepsilon C_y^{(i)})$. Now the induction hypothesis is applicable to both u and v, because $1 \leq |u|, |v| \leq |w| - 2$ and accordingly $|xua| \leq |xw| - 1 \leq |xwy| - 1 = \ell$ and $|vy| \leq |wy| - 2 \leq |xwy| - 2 = \ell - 1$. From this we get $xua \in L_G(B^{(i)})$ and $vy \in L_G(C^{(i)})$, and therefore $xwy \in L_G(B^{(i)}C^{(i)})$. The second case, ${}_x\alpha_y^{(i)} = {}_xB_\varepsilon^{(i)} \cdot a \cdot {}_a C_y^{(i)}$ (9.4b), is symmetric.

For the case (9.4d), assume ${}_x\alpha_y^{(i)} = {}_\varepsilon C_y^{(i)}$. This means that $x \in L_G(B^{(i)})$, which can only be the case for $x \neq \varepsilon$. Furthermore, we know that $w \in L_{G'}({}_\varepsilon C_y^{(i)})$, and

since $|wy| = |xwy| - 1 \leq \ell$, we can conclude from the induction hypothesis that $wy \in L_G(C^{(i)})$. Altogether, this yields $xwy \in L_G(B^{(i)}C^{(i)})$. The last case (9.4c) is again symmetric.

In all four cases, we got $xwy \in L_G(B^{(i)}C^{(i)})$. Since this holds for all $i \in \{1, \ldots, n\}$, it follows that $xwy \in L_G(A)$, which was asserted.

"$\Leftarrow$": Let now $xwy \in L_G(A)$, $w \in \textsc{Odd}$, and we have to prove that $w \in L_{G'}({}_xA_y)$.

Induction basis $|xwy| = 1$: Since $w \in \textsc{Odd}$, the word xwy has $xy = \varepsilon$ and $w = a \in \Sigma$. Now if $xwy = a \in L_G(A)$, then the grammar G' has a corresponding rule (9.5), and $w = a$ is generated from ${}_xA_y$ by this rule.

Induction step: Let $\ell \geq 1$ and assume that the claim holds for all A, x, w and y, where $|xwy| \leq \ell$. Let now $xwy \in L_G(A)$ with $|xwy| = \ell + 1$ and $w \in \textsc{Odd}$. If $|w| = 1$, then the assertion holds because of some rule of type (9.5). Otherwise, $|w| \geq 3$ and there must be a rule $A \to B^{(1)}C^{(1)} \& \ldots \& B^{(n)}C^{(n)} \in R$, such that $xwy \in L_G(B^{(i)}C^{(i)})$ for all $i = 1, \ldots, n$. By the construction, there can be multiple rules in R' that correspond to this rule. We now argue that for every $i = 1, \ldots, n$, there is a suitable conjunct ${}_x\alpha_y^{(i)}$ among (9.4a)–(9.4d) that generates w.

For this, fix i again. Then there must be a partition $w = uv$, where $u, v \in \Sigma^*$, $xu \in L_G(B^{(i)})$ and $vy \in L_G(C^{(i)})$. Note that since w has odd length, either u or v has odd length. Without loss of generality, assume that $|v|$ is odd. Since no nonterminal in G generates the empty word, $|xu| \geq 1$ and thus $|\varepsilon vy| \leq |xu| + |vy| - 1 = |xwy| - 1 = \ell$ and we can use the induction hypothesis to obtain $v \in L_{G'}({}_\varepsilon C_y^{(i)})$. For u there are two cases:

- If $u = \varepsilon$, then $x = xu \in L_G(B^{(i)})$. In this case, the ith conjunct ${}_x\alpha_y^{(i)}$ can be defined as ${}_\varepsilon C_y^{(i)}$, as in (9.4d), and $w = v \in L_{G'}({}_\varepsilon C_y^{(i)}) = L_{G'}({}_x\alpha_y^{(i)})$.
- If $u \neq \varepsilon$, then $u = u'a$ for some $a \in \Sigma$ and $xu'a \in L_G(B^{(i)})$ and thus $u' \in L_{G'}({}_xB_a^{(i)})$ by the induction hypothesis ($|xu'a| = |xu| \leq |xw| - 1 \leq |xwy| - 1 = \ell$ and $|u'|$ is odd). This means that $w = u'av \in L_{G'}({}_xB_a^{(i)} \cdot a \cdot {}_\varepsilon C_y^{(i)})$, so this is a possible conjunct ${}_x\alpha_y^{(i)}$ that generates w.

Now we showed that for every $i = 1, \ldots, n$, there is a legal conjunct ${}_x\alpha_y^{(i)}$ in the respective rule for ${}_xA_y$ in R' that generates w, which implies that $w \in L_{G'}({}_xA_y)$. $\diamond$

The grammar G' constructed above is not yet in odd normal form, because it may contain *unit conjuncts* given by (9.4c) and (9.4d). The known procedure for eliminating such conjuncts [Okh01] is a sequence of substitutions of the bodies of all rules for B inside a rule $A \to \ldots \& B \& \ldots$. Accordingly, once these substitutions are done, the resulting grammar G_1 will contain conjuncts of the form (9.4a) and (9.4b), while all conjuncts of the form (9.4c) and (9.4d) will be eliminated. Then G_1 will be in odd normal form. $\square$

The grammar constructed in Lemma 9.13 generates the words of odd length of the given language. However, it actually encodes the entire information defined in the original grammar, and using the "even rules" allowed in the odd normal form one can generate the original language as it is.

Theorem 9.15. For every conjunctive grammar there exists and can be effectively constructed a conjunctive grammar in odd normal form generating the same language.

Proof. Let $L \subseteq \Sigma^*$ be conjunctive. By Lemma 9.13, there is a conjunctive grammar $G = (\Sigma, N, S, R)$ in odd normal form without even rules, such that for all $a \in \Sigma$,

$$L_G(S) = L \cap \textsc{Odd} \qquad \text{and} \qquad L_G({}_aS_\varepsilon) = a^{-1}L \cap \textsc{Odd}.$$

Then the grammar $G' := (\Sigma, N \cup \{S'\}, S, R')$ with a new nonterminal S' and

$$R' := R \cup \{S' \to \varphi \mid S \to \varphi \in R\} \cup \{S' \to a\,{}_aS_\varepsilon \mid a \in \Sigma\} \cup \{S' \to \varepsilon \mid \text{if } \varepsilon \in L\}$$

is in odd normal form (with even rules) and generates L:

$$\begin{aligned} L_{G'}(S') &= L_G(S) \cup \bigcup_{a\in\Sigma} aL_G({}_aS_\varepsilon) \cup (L \cap \{\varepsilon\}) \\ &= (L \cap \textsc{Odd}) \cup \bigcup_{a\in\Sigma} a(a^{-1}L \cap \textsc{Odd}) \cup (L \cap \{\varepsilon\}) \\ &= (L \cap \textsc{Odd}) \cup \bigcup_{a\in\Sigma} \Big(a(a^{-1}L) \cap a\textsc{Odd}\Big) \cup (L \cap \{\varepsilon\}) \\ &= (L \cap \textsc{Odd}) \cup \Big(L \cap (\textsc{Even} - \{\varepsilon\})\Big) \cup (L \cap \{\varepsilon\}) \\ &= L. \end{aligned}$$

If $L \cap \textsc{Even} = \emptyset$, that is, if L does not contain any words of even length, then $L_{G'}({}_aS_\varepsilon) = \emptyset$ for every $a \in \Sigma$. Unfortunately, checking this property is undecidable in the general case, but if this property holds, then the even rules can be removed without changing the generated language. □

Some corollaries can be inferred from the above theorem. The first one concerns the Greibach normal form for conjunctive grammars. As already mentioned, it is unknown whether every conjunctive grammar can be transformed to that form. However, Theorem 9.15 straightforwardly implies a transformation to Greibach normal form for grammars over a single-letter alphabet.

Corollary 9.16. For every conjunctive grammar over a single-letter alphabet there exists and can be effectively constructed a conjunctive grammar in Greibach normal form generating the same language.

Indeed, since concatenation of languages over $\{a\}$ is commutative, each term BaC in an odd normal form grammar can be equivalently replaced by aBC. The second consequence of Theorem 9.15 is a rather expected closure property of conjunctive languages, which apparently has not yet appeared in the literature:

Theorem 9.17. Conjunctive languages are effectively closed under quotient with letters, and hence under quotient with finite languages.

Proof. Let $L \subseteq \Sigma^*$ be conjunctive and fix $a \in \Sigma$. By Lemma 9.13, there is a conjunctive grammar $G = (\Sigma, N, S, R)$, which contains nonterminal symbols S_a and ${}_bS_a$ for all $b \in \Sigma$ that generate the languages

$$L_G(S_a) = La^{-1} \cap \textsc{Odd} \qquad \text{and} \qquad L_G({}_bS_a) = b^{-1}La^{-1} \cap \textsc{Odd}.$$

Construct the grammar $G' = (\Sigma, N \cup \{S'\}, S', R \cup R')$ with the following additional rules:

$$\begin{array}{ll} S' \to S_a & \\ S' \to b\, {}_bS_a & \text{for all } b \in \Sigma \\ S' \to \varepsilon & \text{if } a \in L(G) \end{array}$$

Then $L(G') = L(G)a^{-1}$. The construction for $a^{-1}L$ is symmetric. The quotient of a conjunctive language with a word $w \in \Sigma^+$ is expressed by applying the above construction $|w|$ times. The quotient with a finite language $F \subset \Sigma^+$ is obtained as $\bigcup_{w \in F} w^{-1}L$. □

9.2.2 Transformation to Restricted Form

The goal of this section is to convert an arbitrary conjunctive grammar to a restricted one generating the same language. The main tool is the identity (9.3) illustrated in Example 9.11, which allows representing a disjunction of two nonterminals, A and B, by a restricted grammar, as long as A and B generate only words of odd length. The latter condition is met for grammars in the odd normal form, which leads to the following general transformation.

Lemma 9.18. For every conjunctive grammar generating a subset of Odd there exists a restricted conjunctive grammar generating the same language.

Proof. By Theorem 9.15, any given grammar can be converted to the odd normal form, and since it generates only words of odd length by assumption, even rules may be removed without changing the language. So assume that the conjunctive grammar G is in odd normal form without even rules. The first goal is to transform it so that for every nonterminal A there is either a unique rule of an arbitrary form $A \to \alpha_1 \& \ldots \& \alpha_n$, or two rules $A \to B \mid C$. During this transformation, the property that $L(A) \subseteq \text{Odd}$ for every nonterminal A should be retained.

Let $A \to r_1 \mid r_2 \mid \ldots \mid r_n$ be the rules for the nonterminal A. Of course, $L(r_i) \subseteq \text{Odd}$ for all i. If $n \geq 2$, then the rules for A are replaced with $A \to B \mid C$, where B and C are two new nonterminals with the rules $B \to r_1 \mid r_2 \mid \ldots \mid r_{n-1}$ and $C \to r_n$. Observe that iterative application of this transformation results in a grammar $G' = (\Sigma, N, S, R)$ that generates the same language as G, still has $L(A) \subseteq \text{Odd}$ for all $A \in N$, and furthermore, for every nonterminal $A \in N$ there is either a unique rule of an arbitrary form, or two rules $A \to B \mid C$.

Next, construct a restricted conjunctive grammar $G'' = (\Sigma, N \cup N' \cup \{O\}, S, R')$, in which $N' = \{A' \mid A \in N\}$ is a disjoint copy of N, O is a new nonterminal, and R' contains the following rules:

$$\begin{array}{ll} A' \to A \mid \varepsilon & \text{for all } A \in N \\ A \to \alpha_1 \& \ldots \& \alpha_n \& O & \text{if } A \to \alpha_1 \& \ldots \& \alpha_n \text{ is the unique rule for } A \text{ in } R \\ A \to B'C' \& O & \text{if } A \to B \mid C \text{ are the rules for } A \text{ in } R \\ O \to OOO & \\ O \to a & \text{for all } a \in \Sigma \end{array}$$

By these rules, $L_{G''}(O) = \textsc{Odd}$, $L_{G''}(A) \subseteq \textsc{Odd}$ and $L_{G''}(A') = L_{G''}(A) \cup \{\varepsilon\}$ for every $A \in N$. Assume now that the nonterminal A has the rule $A \to B'C'\&O$ in R'. Then

$$\begin{aligned} L_{G''}(A) &= \Big(L_{G''}(B) \cup \{\varepsilon\}\Big)\Big(L_{G''}(C) \cup \{\varepsilon\}\Big) \cap \textsc{Odd} \\ &= \Big(L_{G''}(B)L_{G''}(C) \cup L_{G''}(B) \cup L_{G''}(C) \cup \{\varepsilon\}\Big) \cap \textsc{Odd} \\ &= L_{G''}(B) \cup L_{G''}(C). \end{aligned}$$

This means that the rule $A \to B'C'\&O$ can be equivalently replaced by the rules $A \to B|C$. If this replacement is done for all nonterminals $A \in N$, we again arrive at the grammar G' with some additional nonterminals and rules only for these nonterminals. Therefore, $L(G) = L(G') = L(G'')$, and the grammar G'' generates the desired language, which proves the lemma. □

Now consider an arbitrary conjunctive language L, which may contain words of both even and odd length. Lemma 9.18 can be used to construct a restricted conjunctive grammar for $L \cap \textsc{Odd}$. In order to get the whole language L later, it is useful to generate *all* words of even length: this will be a superset of L, which could be intersected with some other languages to obtain L. The inclusion of all words of even length is performed in the following lemma.

Lemma 9.19. For every conjunctive language L, the language $(L \cap \textsc{Odd}) \cup \textsc{Even}$ is generated by a restricted conjunctive grammar.

Proof. Let $G = (\Sigma, N, S, R)$ be a restricted conjunctive grammar generating the language $L \cap \textsc{Odd}$, which is given by Lemma 9.18. Construct a new grammar G' with the following rules, where S', A, B, C and O are new nonterminals:

$S' \to AB\&C \mid \varepsilon$		$C \to CC$	
$A \to S$		$C \to w$	for all $w \in (\Sigma \cap L) \cup \Sigma^2 \cup \Sigma^3$
$A \to a$	for all $a \in \Sigma$	$O \to OOO$	
$B \to O \mid \varepsilon$		$O \to a$	for all $a \in \Sigma$

The concatenation AB generates the following language:

$$\Big((L \cap \textsc{Odd}) \cup \Sigma\Big) \cdot (\textsc{Odd} \cup \{\varepsilon\}) = (\textsc{Even} - \{\varepsilon\}) \cup (L \cap \textsc{Odd}) \cup \Sigma$$

Its intersection with $L_{G'}(C) = (\Sigma^+ - \{a \in \Sigma \mid a \notin L\})$ produces $(\textsc{Even} - \{\varepsilon\}) \cup (L \cap \textsc{Odd})$, and taking the rule $S' \to \varepsilon$ into account, the grammar generates $\textsc{Even} \cup (L \cap \textsc{Odd})$. □

The above construction cannot be used symmetrically to obtain the language $(L \cap \textsc{Even}) \cup \textsc{Odd}$ directly. However, the method of Lemma 9.19 can be elaborated to generate the following superset of L:

Lemma 9.20. For every conjunctive language $L \subseteq \Sigma^*$ and for every symbol $a \in \Sigma$, the language $(L \cap a\textsc{Odd}) \cup \overline{a\textsc{Odd}}$ is generated by some restricted conjunctive grammar.

Proof. Let L be a conjunctive language over Σ and let $a \in \Sigma$. Define $L_a := a(a^{-1}L \cap \text{Odd})$: these are all words of even length in L that start with a, that is, $L_a = L \cap a\text{Odd}$. Define the following three languages:

$$\begin{aligned} L_1 &= (\Sigma - \{a\})\Sigma^* \cup \{\varepsilon\}, \\ L_2 &= L_a \cup \{\varepsilon\}, \\ L_3 &= \text{Odd} \cup \{\varepsilon\}. \end{aligned}$$

Each of these languages has a restricted conjunctive grammar:

- $L_1 = (\Sigma - \{a\})\Sigma^* \cup \{\varepsilon\}$: Observe the restricted conjunctive grammar $G := (\Sigma, \{S, A, X\}, S, R)$ with R containing the rules $S \to AX$, $S \to \varepsilon$, $A \to b$ (for all $b \in \Sigma - \{a\}$), $X \to XX$, $X \to v$ (for all $v \in \Sigma \cup \{\varepsilon\}$). The nonterminals produce the following languages: $L_G(X) = \Sigma^*$, $L_G(A) = \Sigma - \{a\}$ and $L_G(S) = (\Sigma - \{a\})\Sigma^* \cup \{\varepsilon\}$. So the grammar produces the desired language.

- $L_2 = L_a \cup \{\varepsilon\}$: Since L is conjunctive, the language $a^{-1}L \cap \text{Odd}$ is conjunctive by Theorem 9.17, and therefore, by Lemma 9.18, there is a restricted conjunctive grammar generating this language. This grammar can be easily modified to generate L_2.

- $L_3 = \text{Odd} \cup \{\varepsilon\}$: This language is obviously generated by a restricted conjunctive grammar with the rules $S \to O \mid \varepsilon$, $O \to OOO$, $O \to a$ for every $a \in \Sigma$.

Now consider the concatenation of these three languages:

$$\begin{aligned} L_1L_2L_3 &= \Big(\{\varepsilon\} \cup L_a \cup (\Sigma - \{a\})\Sigma^* L_a \cup (\Sigma - \{a\})\Sigma^*\Big) \cdot (\text{Odd} \cup \{\varepsilon\}) \\ &= \Big(\{\varepsilon\} \cup L_a \cup (\Sigma - \{a\})\Sigma^*\Big) \cdot (\text{Odd} \cup \{\varepsilon\}) \\ &= \{\varepsilon\} \cup L_a \cup (\Sigma - \{a\})\Sigma^* \cup \text{Odd} \cup \underbrace{L_a\text{Odd}}_{\subseteq \text{Odd}} \cup \underbrace{(\Sigma - \{a\})\Sigma^*\text{Odd}}_{\subseteq (\Sigma - \{a\})\Sigma^*} \\ &= L_a \cup \text{Odd} \cup (\text{Even} - a\Sigma^*) = L_a \cup \overline{a\text{Odd}} = (L \cap a\text{Odd}) \cup \overline{a\text{Odd}}. \end{aligned}$$

From this, using the grammars for L_1, L_2 and L_3, it is easy to construct a restricted conjunctive grammar for the desired language. □

It remains to intersect the $|\Sigma| + 1$ languages constructed in Lemmas 9.19 and 9.20 to obtain a grammar for any conjunctive language L containing ε, and an extra intersection with Σ^+ settles the case of $\varepsilon \notin L$. This gives the main result of this section:

Theorem 9.21. Every conjunctive language is generated by a restricted conjunctive grammar.

Proof. Let $L \subseteq \Sigma^*$ be any conjunctive language. Then, by Lemmas 9.19 and 9.20, there are restricted conjunctive grammars for the languages $(L \cap \text{Odd}) \cup \text{Even}$ and $(L \cap a\text{Odd}) \cup \overline{a\text{Odd}}$ for each $a \in \Sigma$. The intersection of these languages is

$$\Big((L \cap \text{Odd}) \cup \text{Even}\Big) \cap \bigcap_{a \in \Sigma} \Big((L \cap a\text{Odd}) \cup \overline{a\text{Odd}}\Big) = L \cup \{\varepsilon\}.$$

If $\varepsilon \in L$, this immediately gives a restricted conjunctive grammar for L. Otherwise, if $\varepsilon \notin L$, then a subsequent conjunction with a nonterminal representing Σ^+ yields the required grammar. □

9.2.3 Restricted Conjunctive Grammars Without ε-Rules

The above simulation of an arbitrary conjunctive grammar by a conjunctive grammar with restricted disjunction essentially uses rules of the form $A \to \varepsilon$, known as *ε-rules*. On the other hand, as long as only languages not containing the empty word are concerned, it is known that conjunctive grammars of the general form do not need ε-rules, and a transformation to the binary normal form leads to their elimination [Okh01]. This raises the question of whether restricted conjunctive grammars without ε-rules are as powerful as conjunctive grammars of the general form. Throughout this section, all languages are subsets of Σ^+.

First of all, this stronger restriction on conjunctive grammars still gives a non-trivial family. For instance, the important grammar over a single-letter alphabet given in Example 9.7 is of this form. Grammars for interesting languages over larger alphabets can be constructed as well.

Example 9.22. The following restricted conjunctive grammar generates the set of all (nonempty) palindromes:

$$\begin{array}{lcl@{\qquad}lcl} S & \to & XSX\&T \mid a \mid b \mid aa \mid bb & A & \to & bE \mid a \mid b \\ T & \to & AB\&CD\&XXE & B & \to & Ea \mid a \mid b \\ E & \to & XE \mid a \mid b & C & \to & aE \mid a \mid b \\ X & \to & a \mid b & D & \to & Eb \mid a \mid b \end{array}$$

In particular, $L(E) = \Sigma^+$, $L(A) = b\Sigma^* \cup \{a\}$, $L(B) = \Sigma^* a \cup \{b\}$, $L(C) = a\Sigma^* \cup \{b\}$, $L(D) = \Sigma^* b \cup \{a\}$, and $L(T) = a\Sigma^+ a \cup b\Sigma^+ b$.

Consider the intersection $L(AB) \cap L(CD)$ used in the rule for T:

$$\begin{aligned} (b\Sigma^* \cup \{a\})(\Sigma^* a \cup \{b\}) \cap (a\Sigma^* \cup \{b\})(\Sigma^* b \cup \{a\}) = \\ &= \Big(b\Sigma^* a \cup b\Sigma^* b \cup a\Sigma^* a \cup \{ab\}\Big) \cap \Big(a\Sigma^* b \cup a\Sigma^* a \cup b\Sigma^* b \cup \{ba\}\Big) \\ &= a\Sigma^* a \cup b\Sigma^* b \cup \{ab, ba\}, \end{aligned}$$

and the subsequent intersection with the set of all words of length at least 3 produces the intended language $a\Sigma^+ a \cup b\Sigma^+ b$. Finally, the rule $S \to XSX\&T$ generates the language

$$\{a, b\}S\{a, b\} \cap (a\Sigma^+ a \cup b\Sigma^+ b) = aSa \cup bSb,$$

and hence operates as if two rules $S \to aSa$ and $S \to bSb$. This is enough to generate all palindromes inductively, starting from the base set $\{a, b, aa, bb\}$.

Further investigating this yet more restricted subclass of conjunctive grammars, one can note the following basic properties:

Lemma 9.23. The family of languages generated by restricted conjunctive grammars without ε-rules is closed under union with finite sets, concatenation and intersection.

Proof. The closure under concatenation and under intersection is immediate. For the union with finite sets, let $F \subseteq \Sigma^+$ be finite and let $G = (\Sigma, N, S, R)$ be a restricted conjunctive grammar without ε-rules. The grammar $(\Sigma, N \cup \{S'\}, S', R \cup \{S' \to S\} \cup \{S' \to w \mid w \in F\})$ with the new nonterminal S' is restricted, does not contain ε-rules and generates $L(G) \cup F$. $\square$

The next lemma will be subsumed by the result that all regular languages can be generated (Corollary 9.29), but it will serve as a useful tool.

Lemma 9.24. Any finite or co-finite language can be generated by a restricted conjunctive grammar without ε-rules.

Proof. A finite language $F = \{w_1, \ldots, w_n\}$ is generated by a grammar $S \to w_1 \mid \ldots \mid w_n$.

Let now $L \subseteq \Sigma^+$ be co-finite. Then there is some $k \geq 1$, such that $L \cap \Sigma^k\Sigma^+ = \Sigma^k\Sigma^+$, and there exists a finite set F with $L = \Sigma^k\Sigma^+ \cup F$. Obviously, $\Sigma^k\Sigma^+$ can be generated by a restricted conjunctive grammar without ε-rules. Since L is the union of $\Sigma^k\Sigma^+$ with the finite set F, there is also a restricted conjunctive grammar without ε-rules for L after Lemma 9.23. □

Another language of interest that can be generated by a restricted conjunctive grammar without ε-rules is the earlier mentioned language $\{wcw \mid w \in \{a, b\}^*\}$. The grammar, which will be given in Example 9.28 at the end of this section, can actually be obtained by formally transforming the slightly modified grammar from Example 9.5. This transformation is done in the following theorem, which is applicable to a fairly substantial subfamily of conjunctive grammars including the conjunctive grammars in deterministic Greibach normal form.

Theorem 9.25. Let $G = (\Sigma, N, S, R)$ be a conjunctive grammar without ε-rules, in which there is a disjoint partition of its nonterminals $N = N_L \cup N_R \cup N_S$ into *left*, *right* and *simple* nonterminals, respectively, such that the rules satisfy the following conditions:

- Every $A \in N_S$ has at most one non-terminating rule.
- Each non-terminating rule for $A \in N_L \cup N_R$ has the form $A \to a\alpha_1 \& \ldots \& a\alpha_n$ (if $A \in N_L$) or $A \to \alpha_1 a \& \ldots \& \alpha_n a$ (if $A \in N_R$) for some $a \in \Sigma$, $n \geq 1$ and $\alpha_1, \ldots, \alpha_n \in (\Sigma \cup N)^+$.
- For each pair $(A, a) \in (N_L \cup N_R) \times \Sigma$, there is at most one rule of the form $A \to a\alpha_1 \& \ldots \& a\alpha_n$ (if $A \in N_L$) or $A \to \alpha_1 a \& \ldots \& \alpha_n a$ (if $A \in N_R$).

Then there exists (and can be effectively constructed) a restricted conjunctive grammar without ε-rules that generates the same language.

Before we come to the proof of this theorem, let us see how the grammar in Example 9.5 can be easily transformed to fulfill the requirements:

Example 9.26 (cf. Example 9.5)**.** The conjunctive grammar for the language $\{wcw \mid w \in \{a, b\}^*\}$ that is specified by the following rules

$$\begin{array}{lcl} S & \to & C\&D \\ C & \to & FCF \mid c \\ D & \to & aA\&aD \mid bB\&bD \mid cE \mid c \\ A & \to & aAF \mid bAF \mid cEa \mid ca \\ B & \to & aBF \mid bBF \mid cEb \mid cb \\ E & \to & FE \mid a \mid b \\ F & \to & a \mid b \end{array}$$

satisfies the statement of Theorem 9.25 with $N_L = \{D, A, B\}$, $N_R = \emptyset$ and $N_S = \{S, C, E, F\}$.

Proof of Theorem 9.25. Let $G = (\Sigma, N, S, R)$ be a conjunctive grammar of the stated form. Construct a grammar $G' := (\Sigma, N', S, R')$ such that

$$N' := N \cup \{A_a \mid A \in N_R \cup N_L,\ a \in \Sigma\} \cup \{X_a \mid a \in \Sigma\} \cup \{{}_aX \mid a \in \Sigma\} \cup \{T\} \cup N_2$$

(where N_2 is a set of auxiliary nonterminals that will not be explicitly described) and the set R' contains the following rules. *Simple* nonterminals have the same rules as in R:

$$A \to \alpha_1 \& \ldots \& \alpha_n \qquad \text{for } A \in N_S,\ A \to \alpha_1 \& \ldots \& \alpha_n \in R \tag{9.6a}$$
$$A \to w \qquad \text{for } A \to w \in R,\ w \in \Sigma^+ \tag{9.6b}$$

Left and *right* nonterminals have the following rules:

$$A \to T \& \mathop{\&}_{a \in \Sigma} (X_a \cdot A_a) \qquad \text{for } A \in N_L \tag{9.7a}$$
$$A \to T \& \mathop{\&}_{a \in \Sigma} (A_a \cdot {}_aX) \qquad \text{for } A \in N_R \tag{9.7b}$$
$$A \to w \qquad \text{for } A \to w \in R,\ w \in \Sigma^+ \tag{9.7c}$$
$$A \to w \qquad \text{for } w \in L_G(A),\ |w| = 2, \tag{9.7d}$$

that is, all terminating rules from R are retained, all words of length two are explicitly generated, and the unique non-terminating rule is simulated by the nonterminals A_a using the following rules:

$$A_a \to b \qquad \text{for } A \in N_L \cup N_R,\ a, b \in \Sigma \tag{9.8a}$$
$$A_a \to \alpha_1 \& \ldots \& \alpha_n \qquad \text{for } A \in N_L \cup N_R,\ a \in \Sigma; \quad \text{if } A \to a\alpha_1 \& \ldots \& a\alpha_n \in R \text{ or } A \to \alpha_1 a \& \ldots \& \alpha_n a \in R \tag{9.8b}$$

Additionally, the rules for the nonterminals X_a and ${}_aX$ (for every $a \in \Sigma$) and T are constructed so that $L_{G'}(X_a) = \{a\} \cup (\Sigma - \{a\})\Sigma^*$, $L_{G'}({}_aX) = \{a\} \cup \Sigma^*(\Sigma - \{a\})$ and $L_{G'}(T) = \Sigma^3\Sigma^*$. This can be done by Lemmas 9.23 and 9.24 using the additional nonterminals from the set N_2.

Note that because of the restrictions on G, there is at most one non-terminating rule for every nonterminal $A \in N'$, and thus G' is of the restricted form without ε-rules. The nonterminals of the constructed grammar generate the following languages (which will be proved later):

Claim 9.27. For every $A \in N$ and $a \in \Sigma$ it holds that

1. $L_{G'}(A_a) = \begin{cases} \Sigma \cup \bigcap_{i=1}^{n} L_G(\alpha_i) & \text{if } R \text{ contains a rule } A \to a\alpha_1 \& \ldots \& a\alpha_n \text{ or } A \to \alpha_1 a \& \ldots \& \alpha_n a \text{ with } n \geq 1,\ \alpha_i \in (\Sigma \cup N)^+ \\ \Sigma & \text{otherwise,} \end{cases}$
   provided that $A \notin N_S$.
2. $L_{G'}(A) = L_G(A)$.

Proof of the claim. The main idea of the construction is that of a single rule (9.7a) in G' simulating multiple rules $A \to a\alpha_{a,1}\& \ldots \& a\alpha_{a,n_a}$ in G, for all $a \in \Sigma_A \subseteq \Sigma$, where Σ_A is a set of starting symbols of left rules for $A \in N_L$. The simulation can be illustrated by substituting the intended languages $L_{G'}(T)$, $L_{G'}(X_a)$ and $L_{G'}(A_a)$ into the expression $T \cap \bigcap_{a\in\Sigma} X_a \cdot A_a$. First, under the substitution $X_a = \{a\} \cup (\Sigma - \{a\})\Sigma^*$ and $A_a = L_{A,a}$ for any $L_{A,a}$ with $\Sigma \subseteq L_{A,a} \subseteq \Sigma^+$, the subexpression $\bigcap_{a\in\Sigma} X_a \cdot A_a$ evaluates to

$$\begin{aligned}\bigcap_{a\in\Sigma} \Big[\{a\} \cup (\Sigma - \{a\})\Sigma^*\Big] L_{A,a} &= \bigcap_{a\in\Sigma} \Big[aL_{A,a} \cup (\Sigma - \{a\})\Sigma^* L_{A,a}\Big] \\ &= \bigcap_{a\in\Sigma} \Big[aL_{A,a} \cup (\Sigma - \{a\})\Sigma^+\Big] = \bigcup_{a\in\Sigma} aL_{A,a}.\end{aligned}$$

Then the whole expression $T \cap \bigcap_{a\in\Sigma} X_a \cdot A_a$, under the substitution $T = \Sigma^3\Sigma^*$, $X_a = \{a\} \cup (\Sigma - \{a\})\Sigma^*$ for all $a \in \Sigma$, $A_a = \Sigma \cup \bigcap_{i=1}^{n_a} L_G(\alpha_{a,i})$ for $a \in \Sigma_A$ and $A_a = \Sigma$ for $a \in \Sigma - \Sigma_A$ evaluates to

$$\Sigma^3\Sigma^* \cap \Big((\Sigma - \Sigma_A)\Sigma \cup \bigcup_{a\in\Sigma_A} a\Big(\Sigma \cup \bigcap_{i=1}^{n_a} L_G(\alpha_{a,i})\Big)\Big) = \Big(\bigcup_{a\in\Sigma_A} \bigcap_{i=1}^{n_a} aL_G(\alpha_{a,i})\Big) - \Sigma^2.$$

Words of length two are lost in the process and have to be specially generated by the rules (9.7d).

However, proving that the given languages $L_{G'}(Z)$, for all $Z \in N'$, form a solution of the system of language equations corresponding to G', is not sufficient, as the assumptions of the theorem do not guarantee the uniqueness of a solution of that system. Note that the first part of the claim follows from the second. Hence, it suffices to prove the second part of the claim which is done by showing both inclusions using an induction on the number of derivation steps in the respective grammars.

"$L_{G'}(A) \subseteq L_G(A)$": It is claimed that for every $A \in N$ and $w \in \Sigma^*$, if $A \overset{G'}{\Longrightarrow}{}^{\ell} w$ for some $\ell \geq 1$, then $A \overset{G}{\Longrightarrow}{}^{*} w$.

Induction basis $\ell \leq 2$: The case $\ell = 1$ holds trivially since there is no one-step derivation $A \overset{G'}{\Longrightarrow} w$, because every application of a rule generates a pair of parentheses. A two-step derivation must be of the form $A \overset{G'}{\Longrightarrow} (w\& \ldots \& w) \overset{G'}{\Longrightarrow} w$. So only a rule that directly generates a terminal word (or a conjunction thereof) can be used for the first derivation. For A, such rules are either directly copied from R, as (9.6b) and (9.7c), or are constructed for some $w \in L_G(A)$, as (9.7d), and in both cases $A \overset{G}{\Longrightarrow}{}^{*} w$.

Induction step: Let $\ell \geq 3$ and assume that the assertion holds for derivations in G' of less than ℓ steps. Let $A \overset{G'}{\Longrightarrow}{}^{\ell} w$. Since terminating rules always produce two-step derivations, the first rule that is applied in $A \overset{G'}{\Longrightarrow}{}^{\ell} w$ must be of type (9.6a), (9.7a) or (9.7b).

If $A \in N_S$, then all rules for A have been copied from G (9.6a), 9.6b and the first step of the derivation is an application of some rule $A \to \alpha_1\& \ldots \&\alpha_n \in R \cap R'$, that is, $A \overset{G'}{\Longrightarrow} (\alpha_1\& \ldots \&\alpha_n) \overset{G'}{\Longrightarrow}{}^{\ell-1} w$. Let $\alpha_i = s_{i1} \ldots s_{ik_i}$ with $k_i \geq 0$ and $s_{ij} \in \Sigma \cup N$. Then there must be a factorization $w = w_{i1} \ldots w_{ik_i}$ with $s_{ij} \overset{G'}{\Longrightarrow}{}^{\ell_{ij}} w_{ij}$ for some

$\ell_{ij} < \ell - 1$. If $s_{ij} \in \Sigma$, then $s_{ij} \overset{G}{\Longrightarrow}{}^0 s_{ij} = w_{ij}$, and if $s_{ij} \in N$, then, by the induction hypothesis, $s_{ij} \overset{G}{\Longrightarrow}{}^* w_{ij}$. These derivations can be assembled together to form a derivation $A \overset{G}{\Longrightarrow} (\alpha_1 \& \ldots \& \alpha_n) \overset{G}{\Longrightarrow} \ldots \overset{G}{\Longrightarrow} w$.

If $A \in N_L$, then the derivation of w must begin with a rule $A \to T \& \&_{a\in\Sigma}(X_a \cdot A_a)$ (9.7a). Then $T \overset{G'}{\Longrightarrow}{}^* w$, and thus $|w| \geq 3$. Let $w = bw'$ for some $w' \in \Sigma^*$ and $b \in \Sigma$. Because of the conjunct $X_b \cdot A_b$ for this particular symbol b, there is some $\ell_1 < \ell$, such that $X_b \cdot A_b \overset{G'}{\Longrightarrow}{}^{\ell_1} w$. Since $L_{G'}(X_b) = \{b\} \cup (\Sigma - \{b\})\Sigma^*$, the only prefix of w generated by X_b is b, and hence $X_b \overset{G'}{\Longrightarrow} b$ and $A_b \overset{G'}{\Longrightarrow}{}^{\ell_1 - 1} w'$. Consider the latter derivation; since $|w'| \geq 2$, some rule $A_b \to \alpha_1 \& \ldots \& \alpha_n$ (9.8b) must be used first, and we get the derivation $A_b \overset{G'}{\Longrightarrow} (\alpha_1 \& \ldots \& \alpha_n) \overset{G'}{\Longrightarrow}{}^{l_1 - 2} w'$, where $\alpha_i \in (\Sigma \cup N)^+$. Then, as in the previous case (of $A \in N_S$), it can be proved using the induction hypothesis that $(\alpha_1 \& \ldots \& \alpha_n) \overset{G}{\Longrightarrow}{}^* w'$. Since there must be a rule $A \to b\alpha_1 \& \ldots \& b\alpha_n \in R$, we get $A \overset{G}{\Longrightarrow}{}^* bw' = w$, and the assertion is proved.

The case of $A \in N_R$ is symmetric.

"$L_{G'}(A) \supseteq L_G(A)$": We now show that for every $A \in N$ and $w \in \Sigma^*$, if $A \overset{G}{\Longrightarrow}{}^\ell w$ for some $\ell \geq 1$ then $A \overset{G'}{\Longrightarrow}{}^* w$.

Induction basis $\ell \leq 2$: Similarly to the induction basis for the first inclusion, the assertion is true since all rules for A in R that directly generate terminal words are also present in G'.

Induction step: Assume $A \overset{G}{\Longrightarrow}{}^\ell w$ and assume that the assertion holds for less than ℓ derivation steps. If $A \in N_S$, all rules for A are copied from R to R', and thus the assertion holds with the same argumentation as in the proof for the other inclusion and by the induction hypothesis. Consider the case of $A \in N_L$. Since $\ell \geq 3$, the first rule in the derivation must be $A \to a\alpha_1 \& \ldots \& a\alpha_n$ for some $a \in \Sigma$, and thus $w = aw'$ for some $w' \in \Sigma^+$. If $|w| = 2$, then $A \overset{G'}{\Longrightarrow} w$ by a direct rule (9.7d). Now assume $|w| \geq 3$. Because of $(a\alpha_1 \& \ldots \& a\alpha_n) \overset{G}{\Longrightarrow}{}^{\ell-1} w$ we also get $(\alpha_1 \& \ldots \& \alpha_n) \overset{G}{\Longrightarrow}{}^{\ell-1} w'$ and from this, as we have already argued, $(\alpha_1 \& \ldots \& \alpha_n) \overset{G'}{\Longrightarrow}{}^* w'$. There must be a rule $A_a \to \alpha_1 \& \ldots \& \alpha_n \in R'$ (9.8b) and thus $A_a \overset{G}{\Longrightarrow} (\alpha_1 \& \ldots \& \alpha_n) \overset{G'}{\Longrightarrow}{}^* w'$. Since $a \in L_{G'}(X_a)$, the word $w = aw'$ is in $L_{G'}(X_a A_a)$.

To see that w is also in $L_{G'}(X_b A_b)$ for every $b \in \Sigma - \{a\}$, let $w = aw''c$ for some $c \in \Sigma$. Then $aw'' \in L_{G'}(X_b)$ as a word of length at least 2 starting not from b, and $c \in L_{G'}(A_b)$ by the rule (9.8a). This gives $w \in L_{G'}(X_b A_b)$.

Finally, $w \in L_{G'}(T) = \Sigma^3\Sigma^*$, because $|w| \geq 3$. Putting together all of the above, $w \in L_{G'}(A)$ by the rule $A \to T \& \&_{a\in\Sigma}(X_a \cdot A_a)$, and the assertion is proved. The proof for $A \in L_R$ is done symmetrically. ◇

It follows, in particular, that $L_{G'}(S) = L_G(S)$, and thus the two grammars generate the same language. □

The application of the construction in the proof of Theorem 9.25 to the grammar in Example 9.26 leads to the following grammar:

Example 9.28. The grammar in Example 9.26, generating the language $\{wcw \mid w \in \{a,b\}^*\}$, is transformed to the following restricted conjunctive grammar without ε-rules:

$$
\begin{array}{rcl}
S & \to & C\&D \\
C & \to & FCF \mid c \\
E & \to & FE \mid a \mid b \\
F & \to & a \mid b \\
T & \to & ZZT' \\
T' & \to & ZT' \mid a \mid b \mid c \\
Z & \to & a \mid b \mid c \\
X_a & \to & Y_a \mid a \\
X_b & \to & Y_b \mid b \\
X_c & \to & Y_c \mid c \\
Y_a & \to & Y_aZ \mid b \mid c \\
Y_b & \to & Y_bZ \mid a \mid c \\
Y_c & \to & Y_cZ \mid a \mid b
\end{array}
\qquad
\begin{array}{rcl}
D & \to & T\&X_aD_a\&X_bD_b\&X_cD_c \mid c \mid ca \mid cb \\
D_a & \to & A\&D \mid a \mid b \mid c \\
D_b & \to & B\&D \mid a \mid b \mid c \\
D_c & \to & E \mid a \mid b \mid c \\
A & \to & T\&X_aA_a\&X_bA_b\&X_cA_c \mid ca \\
A_a & \to & AF \mid a \mid b \mid c \\
A_b & \to & AF \mid a \mid b \mid c \\
A_c & \to & Ea \mid a \mid b \mid c \\
B & \to & T\&X_aB_a\&X_bB_b\&X_cB_c \mid cb \\
B_a & \to & BF \mid a \mid b \mid c \\
B_b & \to & BF \mid a \mid b \mid c \\
B_c & \to & Eb \mid a \mid b \mid c
\end{array}
$$

Theorem 9.25 is applicable, in particular, to all LL(1) context-free grammars in Greibach normal form, which are the "simple grammars" studied by Korenjak and Hopcroft [KH66]. As a consequence, all regular languages can be represented:

Corollary 9.29. Every language generated by an LL(1) context-free grammar in Greibach normal form, and in particular every regular language $L \subseteq \Sigma^+$, is restricted conjunctive without ε-rules.

Further Research. The exact expressive power of conjunctive grammars with restricted disjunction and without ε-rules is left as an open question to study. In particular, it would be interesting to investigate it in the case of a single-letter alphabet: perhaps these grammars can generate all conjunctive languages over single-letter alphabets. For larger alphabets, these grammars likely generate a proper subfamily of conjunctive languages.

9.3 Parsing Boolean Languages Over a Single-Letter Alphabet

Having shown some results that may be of interest to the theory of formal languages only, we now turn to complexity questions again. We will give an upper bound on the time needed to determine if a string a^n is generated by a given Boolean grammar G over the alphabet $\{a\}$. Such problems are generally called *membership problems* and in the special case of grammars, this process is also known as *parsing*. For Boolean grammars over arbitrary alphabets, one can adapt the Cocke–Kasami–Younger parsing algorithm for context-free grammars and this adaptation will work in time $O(|G| \cdot n^3)$ [Okh01, Okh04]. Given a Boolean/context-free grammar $G = (\Sigma, N, S, R)$ in binary/Chomsky normal form and an input word $w = a_1 \dots a_n \in \Sigma^*$, this algorithm inductively computes the sets $T_{i,j} = \{A \in N \mid a_{i+1} \dots a_j \in L_G(A)\}$, for all $0 \leq i < j \leq n$. The bottleneck here is the need to compute n^2 unions of the form $\bigcup_{i<k<j} T_{i,k} \times T_{k,j}$, which represent the set of all concatenations BC that generate the corresponding subword $a_{i+1} \dots a_j$.

A more efficient way of calculating these sets via Boolean matrix multiplication was invented by Valiant [Val75], and it can be used to obtain an $O(|G| \cdot \mathrm{BMM}(n))$-time parsing algorithm for context-free and an $O(|G| \cdot \mathrm{BMM}(n) \log n)$-time parsing algorithm for Boolean grammars [Okh10], where $\mathrm{BMM}(n)$ is the complexity of multiplying two $n \times n$ Boolean matrices.

We will show that for Boolean grammars over a single-letter alphabet, similar results can be obtained. A simple implementation of the Cocke–Kasami–Younger algorithm yields an $O(|G| \cdot n^2)$ algorithm. A careful examination shows that the algorithm can be improved by replacing certain operations by Boolean convolution and more specifically, online Boolean convolution. We further show how integer multiplication can be used to obtain fast online Boolean convolution algorithms and thus fast parsing algorithms for Boolean grammars. In particular, we obtain a parsing algorithm for Boolean grammars G in binary normal form over a single-letter alphabet that works in time $O(|G| \cdot \mathrm{M}(n \log n) \log n)$ or $O(\mathrm{BC}(n) \log n)$ where $\mathrm{M}(n)$ and $\mathrm{BC}(n)$ are (upper bounds for) the bit-complexity of multiplying two n-bit integers and of computing the convolution of two n-dimensional Boolean vectors, respectively. We also require some moderate conditions on the growth of $\mathrm{M}(n)$ and $\mathrm{BC}(n)$. For the currently best-known integer multiplication algorithm [Für09], this yields an algorithm with running time $|G|\, n \log^3 n\, 2^{O(\log^* n)} \leq O(|G|\, n \log^4 n)$.

As circuits over the natural numbers with addition as the only arithmetical operation can also be considered as Boolean grammars over single-letter alphabets, this result transfers to the membership problem for such circuits. McKenzie and Wagner showed that this problem is PSPACE-complete [MW07], though it should be noted that they use a different input encoding: They use the binary encoding for the number whose membership is queried whereas here, it is encoded in unary. This means that our result implies that the membership problem for $\{\cup, \cap, {}^{-}, +\}$-circuits is in $\mathrm{DTIME}(2^n\, n^4)$. The same remark applies to the membership problem for systems of equations over sets of natural numbers with union, intersection and addition as considered by Jeż and Okhotin, which is known to be E-complete [JO11a].

9.3.1 Recognition by Convolution

In the case of a single-letter alphabet, a word of length n has only n distinct nonempty subwords, and the basic cubic-time parsing algorithm can be simplified as seen in Algorithm 9.1. The computation that is done in lines 8 to 10 of Algorithm 9.1, essentially determines one output bit of the *Boolean convolution* of the Boolean vectors V_B and V_C. As we will see later, this computation is done quite inefficiently.

Definition 9.30. The *Boolean convolution* maps two Boolean vectors $x = (x_1, \ldots, x_n)$, $y = (y_1, \ldots, y_n) \in \mathbb{B}^n$ to another Boolean vector $z = x \circ y = (z_2, \ldots, z_{2n}) \in \mathbb{B}^{2n-1}$, where

$$z_i = \bigvee \{x_j \wedge y_k \mid 1 \leq j, k \leq n, j + k = n\}.$$

The convolution of two Boolean vectors x and y is *unambiguous*, if for every k with $2 \leq k \leq 2n$, there is at most one pair (i, j), such that $i + j = k$ and $x_i = y_j = 1$.

The special case of unambiguous convolution in the above definition will become important in Section 9.3.2.

Algorithm 9.1: `parse_basic`(G, a^n)

`Input` : Boolean grammar $G = (\{a\}, N, S, R)$ in binary normal form, word a^n for $n \geq 1$
`Output`: Boolean value indicating $a^n \in L(G)$

```
1  for each A ∈ N create a Boolean vector V_A[1..n] initialized to false;
2  let P ⊆ N × N be the set of all pairs of nonterminals occurring (positively or
   negatively) in right-hand sides of rules;
3  for each (B, C) ∈ P create a Boolean variable W_BC;
4  foreach A ∈ N with A → a ∈ R do
5      V_A[1] = true;
6  for i := 2 to n do
7      foreach (B, C) ∈ P do
8          W_BC := false;
9          for j := 1 to i − 1 do
10             W_BC := W_BC ∨ (V_B[j] ∧ V_C[i − j]);
11     foreach A → B_1C_1& ... &B_mC_m&¬D_1E_1& ... &¬D_rE_r&¬ε ∈ R do
12         V_A[i] := V_A[i] ∨ (W_{B_1C_1} ∧ ... ∧ W_{B_kC_k} ∧ ¬W_{D_1E_1} ∧ ... ∧ ¬W_{D_rE_r});
13 return V_S[n];
```

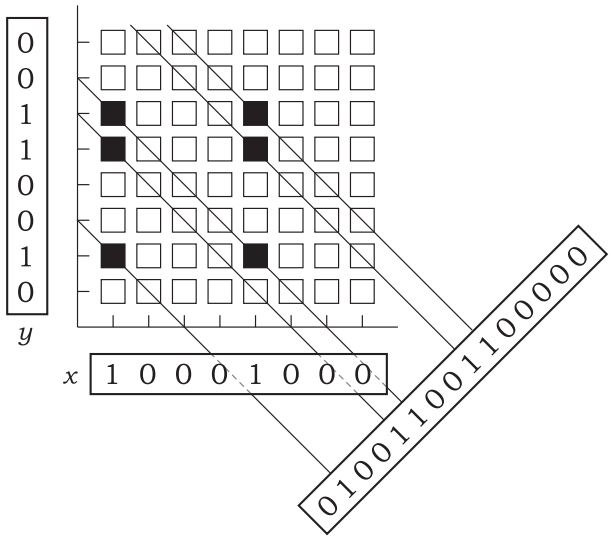


Figure 9.1: Convolution of the Boolean vectors $x = (1, 0, 0, 0, 1, 0, 0, 0)$ and $y = (0, 1, 0, 0, 1, 1, 0, 0)$. Boxes represent conjunctions of bits and they are filled if the conjunction is true. The disjunctions are built along the diagonals.

This definition is illustrated in Figure 9.1, where the axes correspond to the two Boolean vectors, filled and empty boxes represent true and false conjunctions, respectively, and the disjunction of each diagonal of conjunctions produces one component of the convolution. A naïve approach to computing the convolution, as the one implemented in Algorithm 9.1, involves evaluating all $\Theta(n^2)$ Boolean operations, as per the definition. Much more efficient $n \log^{O(1)} n$-time methods based on Fast Fourier Transform were developed in connection with the problem of fast multiplication of integers.

Using these algorithms, one can, for instance, determine the membership of a word a^n in the language defined by a regular expression e over a single-letter alphabet, by calculating the language generated by each subexpression intersected with $\{a\}^{\leq n}$. This works in time $|e| \cdot n \log^{O(1)} n$, where $|e|$ is the number of symbols in the regular expression. Furthermore, the same approach works for determining the membership of a number n in a circuit over sets of natural numbers with operations $\{\cup, \cap, {}^-, +\}$.

However, this method of evaluating subexpressions one by one is not directly applicable to Boolean grammars, which usually include circularities in the definition. In general, the membership of a word a^n in a language $L_G(A)$ depends upon the membership of all words $a^1, \ldots, a^{n-1}$ in the languages generated by all nonterminals of the grammar $G = (\{a\}, N, S, R)$, and there is no known way to compute the membership of a^n in $L_G(A)$ without first computing the membership of all shorter words in all languages. To be precise, the membership of a^n in $L_G(A)$ is a function of the membership of a^n in $L_G(BC)$, for all $B, C \in N$, and the latter is one component of the convolution computed in the lines 8–10 of Algorithm 9.1. Since the Boolean vectors being convolved depend on the previously calculated components of the convolution, one must use the online variant of Boolean convolution, defined as follows.

Definition 9.31. Let $x = (x_1, \ldots, x_n)$, $y = (y_1, \ldots, y_n) \in \mathbb{B}^n$ be two Boolean vectors. An *online convolution algorithm*, which computes their convolution $z = x \circ y = (z_2, \ldots, z_{2n})$, receives x and y component by component, and writes each z_i before reading any input x_j, y_j for $j \geq i$.

Fischer and Stockmeyer [FS74] showed how to transform ordinary convolution algorithms into their online variants with not much overhead in the computation. Though they are mainly concerned with integer multiplication, it is also shown [FS74, Sect. 4.1] how this result can be extended to so-called *generalized linear products* defined by Fischer and Paterson [FP74]. Since Boolean convolution is such a generalized linear product, the following holds.

Theorem 9.32 (Fischer, Stockmeyer [FS74])**.** Consider any algorithm computing the Boolean convolution of two Boolean vectors of length n in time $C(n)$, and assume that C is monotone and satisfies $2\,C(n) \leq C(2n) \leq c \cdot C(n)$ for some c. Then there is an online convolution algorithm that runs in time $O(C(n) \cdot \log n)$.

Since Fischer and Stockmeyer do not explicitly consider the case of Boolean convolution, we now give a simplified version of their multiplication algorithm, hereby applied to Boolean convolution.

Proof. Algorithm 9.2 solves the stated problem, where `conv()` is a subroutine that computes (the standard offline) convolution of two Boolean vectors of length n in time $C(n)$.

Algorithm 9.2: `oconv`(x, y)

`Input` : Boolean vectors $(x_1, x_2, \ldots, x_n)$, $(y_1, y_2, \ldots, y_n) \in \mathbb{B}^n$, access to higher indices yields false
`Output`: Boolean vector $(z_2, z_3, \ldots, z_{2n}) \in \mathbb{B}^{2n-1}$

```
1 t_2 := t_3 := ··· := t_{2n} := false;
2 for i := 2 to 2n do
3     foreach k ∈ {2^r | 1 ≤ 2^r < i, 2^r divides i} do
4         (u_2, u_3, ..., u_{2k}) := conv((x_k, ..., x_{2k-1}), (y_{i-k}, ..., y_{i-1}));
5         (u'_2, u'_3, ..., u'_{2k}) := conv((y_k, ..., y_{2k-1}), (x_{i-k}, ..., x_{i-1}));
6         (t_i, ..., t_{i+2k-2}) := (t_i, ..., t_{i+2k-2}) ∨ (u_2, ..., u_{2k}) ∨ (u'_2, ..., u'_{2k})
7     output z_i := t_i;
```

Figure 9.2 illustrates how the convolution of the given vectors is decomposed into convolutions of subvectors of size 1×1, 2×2, 4×4, …, $2^j \times 2^j$, each represented as a box. The loop in line 3 iterates over all powers of 2 strictly less than i that are divisors of i, and calculates two convolutions of each size. The filled squares in the figure illustrate the convolutions calculated for $i = 20$, when the loop is executed for $k = 1, 2, 4$.

Correctness. We first verify the online condition. Because of line 7, we only have to check that in each iteration of the main loop, the variables $x_i, \ldots, x_n$ and $y_i, \ldots, y_n$ are not accessed. Choose any iteration of the main loop and of the inner loop with $k = 2^r$ and observe line 4. Note that since $2^r < i$ and $2^r | i$ it holds that $k = 2^r \leq \frac{i}{2}$ and thus $2k - 1 \leq i - 1$. This means that both arguments of the function `conv`() only access the input up to bit $i - 1$. Because of symmetry, this is also true for line 5 and thus the online condition is fulfilled.

In order to show that the algorithm indeed computes the convolution we consider two directions. For the first, we prove that for every $j \in \{2, 3, \ldots, 2n\}$ such that the output bit j of the convolution of $(x_1, \ldots, x_n)$ and $(y_1, \ldots, y_n)$ is true, the algorithm sets z_j to true. In this case, there must be some $1 \leq s, \ell \leq n$ such that $s + \ell = j$ and $x_s = y_\ell =$ true. Because of the symmetry of the algorithm with respect to x and y we can assume that $s \leq \ell$. Let $r \geq 0$ such that $2^r \leq s < 2^{r+1}$ and $m \geq 2$ such that $(m-1) \cdot 2^r \leq \ell < m \cdot 2^r$. Let $i := m \cdot 2^r$ and $k := 2^r$ and consider the respective iterations of the algorithm. Of course, we need to show that these iterations exist. Since $r \geq 0$ and $m \geq 2$, we have $i \geq 2$. Furthermore, it holds that $i = m \cdot 2^r = 2^r + (m-1) \cdot 2^r \leq s + \ell = j \leq 2n$. This shows that such an iteration of the main loop exists and that its value for i is at most j, which means that the change in the variable t_j is still taken into account. The conditions $1 \leq 2^r < i$ and $2^r | i$ are obviously fulfilled since $i = m \cdot 2^r$ with $m \geq 2$, so the iteration with our chosen value for k also exists. Observe line 4 and note that the bits x_s and y_ℓ are mentioned in the vectors $(x_k, \ldots, x_{2k-1})$ and $(y_{i-k}, \ldots, y_{i-1})$ since $k = 2^r \leq s < 2^{r+1} = 2k$ and $i - k = m \cdot 2^r - 2^r = (m-1) \cdot 2^r \leq \ell < m \cdot 2^r = i$. Because $x_s = y_\ell =$ true, the output bit $s - k + 1 + \ell - (i - k) + 1 = s + \ell - i + 2 = j - i + 2$ of the internal convolution routine is set to true, stored in u_{j-i+2} and finally $t_{(j-i+2)-2+i} = t_j$ is set to true in line 6. As already mentioned, this value is finally assigned to z_j in the iteration with $i = j$.

For the other direction, we show that for every $j \in \{2, 3, \ldots, 2n\}$ such that z_j assumes the value true in the algorithm, the output bit j of the convolution of $(x_1, \ldots, x_n)$ and $(y_1, \ldots, y_n)$ is true. So let $j \in \{2, \ldots, 2n\}$ and $z_j =$ true. Let i and $k = 2^r$ be the values of the loop variables where t_j is set to true for the first time. Without loss

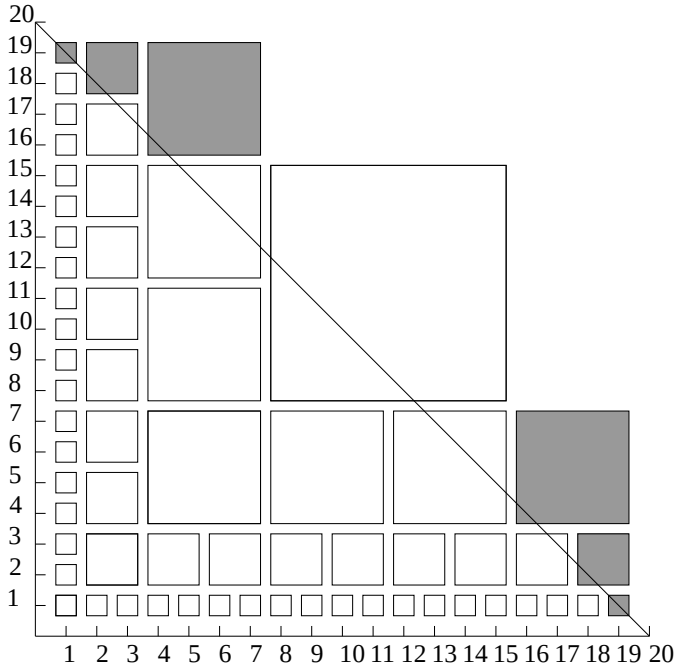


Figure 9.2: Scheme of convolutions computed by Algorithm 9.2 at iteration $i = 20$ (Proof of Theorem 9.32). The axes represent the input Boolean vectors and the squares stand for convolutions while filled squares are convolutions that are computed in iteration $i = 20$. The diagonal crosses all pairs of input bits that contribute to the output bit $i = 20$.

of generality, assume that this truth value originates in $(u_2, u_3, \ldots, u_{2k})$ and thus is computed by the convolution routine in line 4. In order for t_j to be true, there must be $1 \leq s', \ell' \leq k$ with $s' + \ell' = j - i + 2$ such that $x_{k+s'-1} = y_{i-k+\ell'-1} = \text{true}$. Since $k + s' - 1 + i - k + \ell' - 1 = i + j - i + 2 - 1 - 1 = j$, this means that the output bit j of the convolution of $(x_1, \ldots, x_n)$ and $(y_1, \ldots, y_n)$ is true.

Time Complexity. Note that the time needed for any operation apart from computing the convolutions in lines 4 and 5 can be neglected. We start by counting how often these convolutions are computed. For each $r \geq 0$, there are at most $\frac{2n}{2^r} - 1$ values for i such that $2^r < i \leq 2n$ and $2^r | i$. Thus, for each fixed value for $k = 2^r$, the lines 4 and 5 are each reached at most $\frac{2n}{k}$ times. This means that there are at most $4n\,2^{-r}$ calls to a convolution procedure of size 2^r and the possible values for r range from 0 to $\lfloor \log 2n \rfloor$, which means that the overall time taken for the convolutions is at most

$$\begin{aligned}
\sum_{r=0}^{\lfloor \log 2n \rfloor} 4n\,2^{-r} C(2^r) &\leq \sum_{r=0}^{\lfloor \log 2n \rfloor} 4n\,2^{-r}\,2^{r-\lfloor \log 2n \rfloor} C(2^{\lfloor \log 2n \rfloor}) \\
&\leq \sum_{r=0}^{\lfloor \log 2n \rfloor} 4\,C(2n) \\
&\leq 4(2 + \log n)c \cdot C(n)
\end{aligned}$$

for some constant c. Here, the first inequality holds by using $2\,C(n) \leq C(2n)$ multiple times, the second inequality is due to C being monotone and the third inequality is true because $C(2n) \leq c \cdot C(n)$ for some c.

This shows that the algorithm runs in time $O(C(n) \cdot \log n)$. □

Using this result, we can change Algorithm 9.1 from page 161 to directly use online convolution and can concentrate on finding a good algorithm for ordinary convolution.

Algorithm 9.3: `parse`(G, a^n)

`Input` : Boolean grammar $G = (\{a\}, N, S, R)$ in binary normal form, word a^n for $n \geq 1$
`Output`: Boolean value indicating $a^n \in L(G)$

```
1  for each A ∈ N create a Boolean vector V_A[1..n] initialized to false;
2  let P ⊆ N × N be the set of all pairs of nonterminals occurring (positively or
   negatively) in right-hand sides of rules;
3  for each (B, C) ∈ P create a Boolean variable W_BC;
4  for each (B, C) ∈ P run a parallel instance of an online convolution algorithm
   oconv_BC(V_B[1..n], V_C[1..n]) whose output is accessed as an array oconv_BC[2..2n];
5  foreach A ∈ N with A → a ∈ R do
6      V_A[1] := true
7  for i := 2 to n do
8      feed all V_A[i − 1], A ∈ N to the convolution algorithms;
9      foreach (B, C) ∈ P do
10         W_BC := oconv_BC[i];
11     foreach A → B_1C_1& ... &B_kC_k&¬D_1E_1& ... &¬D_rE_r&¬ε ∈ R do
12         V_A[i] := V_A[i] ∨ (W_B1C1 ∧ ... ∧ W_BkCk ∧ ¬W_D1E1 ∧ ... ∧ ¬W_DrEr);
13 return V_S[n];
```

Lemma 9.33. Let $\mathrm{BC}^{\mathrm{o}}(n) \geq n$ be the complexity of computing an n-bit online Boolean convolution. The problem whether a given word a^n is generated by a given Boolean grammar G in binary normal form can be solved in time $O(|G| \cdot \mathrm{BC}^{\mathrm{o}}(n))$.

Proof. We show that Algorithm 9.3 solves the stated problem.

Correctness. We first show that the algorithms $\texttt{oconv}_{BC}$ are used in a way such that they always have enough bits of the input available to produce the requested output bits.

Let $2 \leq i \leq n$ and $(B, C) \in P$. Because of line 8, the bit $\texttt{oconv}_{BC}[i]$ is requested after the bits $V_B[1..(i-1)]$ and $V_C[1..(i-1)]$ have been fed to the algorithm and thus, the online convolution algorithm can correctly produce the requested output bit.

It remains to show that this algorithm does the same as Algorithm 9.1 from page 161. Note that after the initialization, the replacement of lines 7 to 10 by the lines 8 to 10 is the only difference in the algorithms. Since $\texttt{oconv}_{BC}$ computes the convolution, i.e.

$$W_{BC} = \texttt{oconv}_{BC}[i] = \bigvee_{j=1}^{i-1} V_B[j] \wedge V_C[i-j],$$

both algorithms obviously compute the same result.

Time Complexity. Because we run one online convolution algorithm for each pair $(B, C) \in P$, we get $O(|G| \cdot \mathrm{BC}^{\mathrm{o}}(n))$ for the online convolutions alone. All other operations need $O(|G| \cdot n)$ time, and since $\mathrm{BC}^{\mathrm{o}}(n) \geq n$, we get the stated complexity. □

9.3.2 Boolean Convolution

We have seen how parsing Boolean grammars over single-letter alphabets can been reduced to online Boolean convolution, which in turn was reduced to ordinary Boolean convolution. We now apply the final reduction, and show how to use an arbitrary integer multiplication algorithm to compute Boolean convolutions. As a consequence, progress in algorithms or implementations for integer multiplication will also improve the complexity of parsing Boolean grammars over single-letter alphabets.

Lemma 9.34. Let $\mathrm{M}(n)$ be the time complexity of multiplying two n-bit integers. The Boolean convolution of two Boolean vectors of length n can be computed in time $O(\mathrm{M}(n \log n))$ and in time $O(\mathrm{M}(n))$ if the convolution is unambiguous.

Proof. In the following, we interpret $\mathbb{B} = \{0, 1\}$ as a subset of the integers. Let $x = (x_0, x_1, \ldots, x_{n-1})$, $y = (y_0, y_1, \ldots, y_{n-1}) \in \mathbb{B}^n$. We begin with the unambiguous case. Define the n-bit numbers

$$a = \sum_{i=0}^{n-1} x_i 2^i \qquad \text{and} \qquad b = \sum_{j=0}^{n-1} y_j 2^j.$$

The product of these numbers is

$$a \cdot b = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} x_i y_j 2^{i+j} = \sum_{k=0}^{2n-2} 2^k \sum_{i+j=k} x_i y_j = \sum_{k=0}^{2n-2} 2^k \sum_{i=0}^{k} x_i y_{k-i}$$

and since the convolution is unambiguous, the coefficient of each 2^k is either zero or one. Thus the Boolean vector of the convolution coincides with the binary representation of the product.

This is not the case anymore if the convolution is not unambiguous, but there, carries can be avoided by padding in the following way: For

$$a = \sum_{i=0}^{n-1} x_i 2^{i\lceil \log n \rceil} \qquad \text{and} \qquad b = \sum_{j=0}^{n-1} y_j 2^{j\lceil \log n \rceil}$$

we get

$$a \cdot b = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} x_i y_j 2^{(i+j)\lceil \log n \rceil} = \sum_{k=0}^{2n-2} 2^{k\lceil \log n \rceil} \sum_{i+j=k} x_i y_j. = \sum_{k=0}^{2n-2} 2^{k\lceil \log n \rceil} \sum_{i=0}^{k} x_i y_{k-i}.$$

Since for any k, it holds that $\sum_{i+j=k} x_i y_j \leq n \leq 2^{\lceil \log n \rceil}$, there will be no carry between blocks of $\lceil \log n \rceil$ bits, and thus the Boolean vector of the convolution can be extracted from the binary representation of the product. As the multiplication is done on integers with binary length $n\lceil \log n \rceil$, the assertion follows. □

Applying the currently best known upper bound for integer multiplication due to Fürer [Für09] leads to the following result.

Proposition 9.35. The Boolean convolution of two Boolean vectors of length n can be computed in time $n \log^2 n \cdot 2^{O(\log^* n)}$, and in time $n \log n \cdot 2^{O(\log^* n)}$ if the convolution is unambiguous.

Proof. We apply Fürer's algorithm for multiplication with time complexity $n \log n \cdot 2^{O(\log^* n)}$. The unambiguous case is obvious and in the general case we get a time complexity of $O((n \log n) \log(n \log n) 2^{O(\log^*(n \log n))}) = n \log^2 n 2^{O(\log^* n)}$. □

9.3.3 The Resulting Algorithm

Combining the algorithms presented above we arrive at the following upper bound on the complexity of the membership problem for Boolean grammars over a single-letter alphabet. Let us first define a class of Boolean grammars where the resulting algorithm is faster.

Definition 9.36 ([Okh08])**.** The concatenation of two languages, K and L, is said to be *unambiguous*, if every string $w \in K \cdot L$ has a unique factorization $w = uv$ with $u \in K$ and $v \in L$.

A Boolean grammar $G = (\Sigma, N, S, R)$ that generates a language has *unambiguous concatenation*, if all concatenations in the corresponding system of language equations (9.2) are unambiguous under the substitution $A = L_G(A)$ for $A \in N$.

Finally, G is *unambiguous* if it has unambiguous concatenation and furthermore, every union in the system (9.2) is disjoint under the substitution $A = L_G(A)$ for $A \in N$.

Note that if a Boolean grammar G has unambiguous concatenation, then the grammar that is obtained by the transformation to the binary normal form is unambiguous [Okh08].

Theorem 9.37. Consider any algorithm for multiplying two n-bit integers in time at most $M(n)$, and assume that M is monotone and satisfies $2\,M(n) \leq M(2n) \leq c \cdot M(n)$ for some c.

Then there is an algorithm for testing whether a given word a^n is generated by a given Boolean grammar G in binary normal form, which works in time $O(|G| \cdot M(n \log n) \cdot \log n)$. If the grammar is known to be unambiguous, a variant of the algorithm works in time $O(|G| \cdot M(n) \cdot \log n)$.

The currently best known value for $M(n)$ yields the complexity $|G| \cdot n \log^3 n \cdot 2^{O(\log^* n)}$ and $|G| \cdot n \log^2 n \cdot 2^{O(\log^* n)}$, respectively.

Proof. Note that if the grammar is known to be unambiguous, then all convolutions will be unambiguous. By Lemma 9.34, there is a constant c' such that Boolean convolution can be computed in time $c'\,M(n \log n)$ and unambiguous Boolean convolution in time $c'\,M(n)$. Note that both $c'\,M(n \log n)$ and $c'\,M(n)$ fulfill the preconditions of Theorem 9.32. Thus, these algorithms can be converted to online variants with complexity $C^{\mathrm{o}}(n) = O(M(n \log n) \log n)$ and $O(M(n) \log n)$, respectively. Finally, from Lemma 9.33, we obtain the assertions. Using Fürer's result $M(n) = n \log n \cdot 2^{O(\log^* n)}$ [Für09], the explicit bounds follow (note that this function fulfills the requirements of the theorem). □

Finally, consider Algorithm 9.4, which solves the parsing problem and is a combination of Algorithm 9.3 from page 165, and Algorithm 9.2 from page 163, and can be directly implemented given a procedure `conv` for offline Boolean convolution.

Algorithm 9.4: `parse_complete`(G, a^n)

`Input` : Boolean grammar $G = (\{a\}, N, S, R)$ in binary normal form, word a^n for $n \geq 1$
`Output`: Boolean value indicating $a^n \in L(G)$

```
1  for each A ∈ N create a Boolean vector V_A[1..n] initialized to false;
2  let P ⊆ N × N be the set of all pairs of nonterminals occurring (positively or
   negatively) in right-hand sides of rules;
3  for each (B, C) ∈ P create a Boolean vector W_BC[1..n] initialized to false;
4  foreach A ∈ N with A → a ∈ R do
5      V_A[1] := true
6  for i := 2 to n do
7      foreach k ∈ {2^j | 1 ≤ 2^j < i, 2^j divides i} do
8          foreach (B, C) ∈ P do
9              U := conv(V_B[k..2k − 1], V_C[i − k..i − 1]);
10             U' := conv(V_B[i − k..i − 1], V_C[k..2k − 1]);
11             W_BC[i..i + 2k − 1] := W_BC[i..i + 2k − 1] ∨ U ∨ U';
12     foreach A → B_1C_1& ... &B_kC_k&¬D_1E_1& ... &¬D_rE_r&¬ε ∈ R do
13         V_A[i] := V_A[i] ∨ (W_{B_1C_1}[i] ∧ ... ∧ W_{B_kC_k}[i] ∧ ¬W_{D_1E_1}[i] ∧ ... ∧ ¬W_{D_rE_r}[i]);
14 return V_S[n];
```

Chapter 10

Circuits over Sets of Natural Numbers

10.1 Introduction

In 1973, Stockmeyer and Meyer [SM73] defined and investigated equivalence problems for *expressions over sets of natural numbers* (they called them *integer expressions*). They considered expressions that can be built up from single natural numbers by using Boolean operations ($^-$, $\cup$, $\cap$), addition ($+$), and multiplication ($\times$). The *equivalence problem for expressions over sets of natural numbers* is the question of whether two given such expressions describe the same set of natural numbers. Restricting the set of allowed operations results in equivalence problems of different complexities. Stockmeyer and Meyer [SM73] showed that the equivalence test for expressions with the operations $\{^-, \cup, \cap, +\}$ is PSPACE-complete, and that this problem becomes Π_2^{P}-complete if one restricts to operations from $\{\cup, +\}$.

We continue these investigations and study variations of the equivalence problems for expressions and related satisfiability problems in a systematic way. Despite their basic definition, expressions over sets of natural numbers are powerful enough to describe highly non-trivial sets. For instance, the set of primes can be described as

$$\textsc{Primes} = \overline{\overline{0 \cup 1} \times \overline{0 \cup 1}} \ \cap \ \overline{0 \cup 1}.$$

This can be easily verified: The complement of $\{0, 1\}$ multiplied with itself yields all composite numbers. Taking its complement gives the set consisting of 0, 1, and all primes. The intersection with $\overline{0 \cup 1}$ yields the set of primes. Using equivalence problems for these expressions, one can express some of the most prominent, unsolved problems in mathematics.

In 1742, Christian Goldbach stated his famous conjecture as a footnote in a letter to Leonhard Euler. This conjecture is nowadays specified as follows:

Conjecture 10.1 (Goldbach's Conjecture)**.** Every even integer greater than two is the sum of two primes.

The following integer expression describes exactly the set of integers that are counterexamples for Goldbach's conjecture.

$$\textsc{Counterexamples} = (2 \ \times \ \overline{0 \cup 1}) \ \cap \ \overline{\textsc{Primes} + \textsc{Primes}}$$

The left set of the intersection is the set of even integers greater than two, while the right set consists of those integers that are not a sum of two primes. Goldbach's conjecture is true if and only if the set of counterexamples is empty. Therefore,

$$\text{Goldbach's conjecture holds} \iff \textsc{Counterexamples} \text{ is equivalent to } 0 \cap \overline{0}.$$

Hence, a terminating algorithm for the equivalence problem would immediately solve Goldbach's conjecture and thus, this problem is likely undecidable. Indeed, the decidability of the general equivalence problem will be one of our open questions.

Circuits Over Sets of Natural Numbers. Stockmeyer and Meyer's [SM73] motivation for the study of equivalence problems for expressions over sets of natural numbers originated from equivalence problems for Kleene's regular expressions [MS72]. Since then, several variants and generalizations of expressions over sets of natural numbers have been studied. Researchers were also interested in *circuits over sets of natural numbers* which were introduced by Wagner [Wag84]. The latter represent expressions in a succinct way and so yield problems of higher complexity.

Wagner [Wag84], Yang [Yan00], and McKenzie and Wagner [MW07] studied the complexity of *membership problems* for formulas and circuits over natural numbers: Here, for a given circuit C and a number n, one has to decide whether n belongs to the set that is described by C. It was shown that the complexity of the membership problem heavily depends on the operations allowed. It turned out that these problems are complete for various classes between L and NEXP. Breunig [Bre07] studied membership problems for formulas and circuits over $\mathbb{Z}^+$, the positive integers, while Travers [Tra06] studied the variant for $\mathbb{Z}$, the integers. Furthermore, Düntsch and Pratt-Hartmann investigated which kind of sets are representable as such expressions [PD09] and also addressed questions regarding the structure of general complex algebras over the natural numbers [DP09].

The extension from expressions to circuits can be continued to allow cycles in the circuits. This can be modeled by *systems of equations over sets of natural numbers*, which were investigated by Jeż and Okhotin [JO08, JO10, JO11a, JO11b]. The motivation for these systems stems from the correspondence to formal languages over a single-letter alphabet: Every word over a single-letter alphabet can be completely characterized by its length and concatenation is nothing else that addition of word lengths. Systems of equations over sets of natural numbers with operations $\{\cup, +\}$ then coincide with context-free grammars over a single-letter alphabet [GR62] and the extension by the operations $\cap$ and $^-$ correspond to conjunctive and Boolean grammars [Okh01, Okh04]. We refer to chapter 9 for details about these grammars in general and the special case of a single-letter alphabet.

Equivalence and Satisfiability Problems. In this chapter, we study equivalence and satisfiability problems for circuits over sets of natural numbers and for most of these problems we can precisely characterize their complexity. The satisfiability problems are generalizations of the membership problems studied by McKenzie and Wagner [MW07]. In contrast to membership problems, here a circuit can contain variable but also constant input gates. Hence, solving a satisfiability problem is at least as hard as solving the corresponding membership problem.

Notice that the domain of the input variables is unbounded, hence it is not a priori clear that our satisfiability problems are decidable. Indeed, we can prove that some satisfiability problems are undecidable: We show that the problem of solving Diophantine equations, which was proven to be undecidable by Matiyasevich [DPR61, Mat70], can be reduced to $\mathrm{SC}(\cap, +, \times)$, the problem of testing satisfiability for $\{\cap, +, \times\}$-circuits.

It turns out that our results for equivalence problems also have consequences for the known results about membership problems. In fact, our upper bound for the equivalence problem for $\{^-, \cup, \cap, +, \times\}$-circuits yields an improved upper bound for the membership problem for $\{^-, \cup, \cap, +, \times\}$-circuits. This is the first nontrivial upper bound for $\mathrm{MC}(^-, \cup, \cap, +, \times)$, the most general membership problem.

Our main open question for equivalence problems is whether the unrestricted version of the equivalence problem, $\mathrm{EC}(^-, \cup, \cap, +, \times)$, is decidable or not. While we can show that this problem is equivalent to the corresponding membership problem, the upper bound we provide is not a decidable upper bound. So if one proves that $\mathrm{EC}(^-, \cup, \cap, +, \times)$ is undecidable, then it follows that $\mathrm{MC}(^-, \cup, \cap, +, \times)$ also is undecidable.

Similar questions remain open for satisfiability problems. It is unclear whether $\mathrm{SC}(^-, \cup, \cap, \times)$, the satisfiability problem for $\{^-, \cup, \cap, \times\}$-circuits, is decidable. A further open question is to find a better lower bound for the satisfiability problem for $\{\times\}$-circuits. We prove this problem to be in $\mathrm{UP} \cap \mathrm{coUP}$.

The results obtained in this chapter are summarized (together with known results) in Table 10.1 on page 199.

10.2 Definitions

We define the circuit model and related decision problems followed by some examples of circuits. At the end of this section, we will show a lemma providing some general constructions related to circuits.

Definition 10.2. A *circuit* $C = (V, E, g_C)$ is a finite, non-empty, directed, acyclic multigraph (V, E) with a specified node $g_C \in V$. The graph does not have to be connected, and $V = \{1, 2, \ldots, n\}$ for some $n \in \mathbb{N}$. Moreover, the nodes in the graph (V, E) are topologically ordered, i. e., for all $v_1, v_2 \in V$, if $v_1 < v_2$, then there is no path from v_2 to v_1. The nodes in V are also called *gates*. Nodes with indegree 0 are called *input gates* and g_C is called the *output gate*. If in a circuit there is an edge from gate u to gate v, then we say that u is a *direct predecessor* of v and v is the *direct successor* of u. If there is a path from u to v but u is not a direct predecessor of v, then u is an *indirect predecessor* of v and v is an *indirect successor* of u.

Definition 10.3. Let $\mathcal{O} \subseteq \{^-, \cup, \cap, +, \times\}$. An *$\mathcal{O}$-circuit* $C = (V, E, g_C, \alpha)$ is a circuit (V, E, g_C) whose gates are labeled by the labeling function $\alpha \colon V \to \mathcal{O} \cup \mathbb{N}$ such that the following holds: Each gate has an indegree in $\{0, 1, 2\}$, gates with indegree 0 have labels from $\mathbb{N}$, gates with indegree 1 have label $^-$, and gates with indegree 2 have labels from $\{\cup, \cap, +, \times\}$. If each gate that is not an input gate has outdegree at most one, $\mathcal{O}$ is called a *$\mathcal{O}$-formula* or *$\mathcal{O}$-expression*. Each gate in C computes a subset of $\mathbb{N}$ defined by the mapping $\mathrm{I}_C \colon V \to 2^{\mathbb{N}}$. This mapping is inductively defined for each gate $g \in V$ with direct predecessors g_1 and g_2 (if any) as follows:

- If g is an input gate, then $\mathrm{I}(g) := \{\alpha(g)\}$.
- If g has label $^-$, then $\mathrm{I}(g) := \mathbb{N} - \mathrm{I}(g_1)$.
- If g has label $\circ \in \{\cup, \cap, +, \times\}$, then we define $\mathrm{I}(g) := \mathrm{I}(g_1) \circ \mathrm{I}(g_2)$.

If the circuit C is clear from the context, we set $\mathrm{I} := \mathrm{I}_C$. We define $\mathrm{I}(C) := \mathrm{I}_C(g_C)$, the set computed by the circuit C.

For the satisfiability problems studied in section 10.4, we define circuits that, apart from inputs that are assigned numbers, also have unassigned inputs.

Definition 10.4. For $\mathcal{O} \subseteq \{^-, \cup, \cap, +, \times\}$ an *$\mathcal{O}$-circuit $C = (V, E, g_C, \alpha)$ with n unassigned inputs*, $n \geq 0$ is a circuit (V, E, g_C) whose gates are labeled by the labeling function $\alpha\colon V \to \mathcal{O} \cup \mathbb{N} \cup \{\star\}$ such that the following holds: If $u_1 < u_2 < \cdots < u_n$ are the nodes that are labeled by $\star$ and $x_1, \ldots, x_n \in \mathbb{N}$ then $C(x_1, \ldots, x_n) := (V, E, g_C, \alpha')$ is an $\mathcal{O}$-circuit where $\alpha'(u_i) = x_i$ for all $1 \leq i \leq n$ and $\alpha'(v) = \alpha(v)$ for all other gates. This means that C becomes an $\mathcal{O}$-circuit by assigning its inputs. The input gates with label $\star$ are called *unassigned* (or *variable*) input gates (by the definition only input gates can have the label $\star$), the other input gates are called *assigned* (or *constant*) inputs. Each $\mathcal{O}$-circuit C is also an $\mathcal{O}$-circuit with 0 unassigned inputs (all inputs are assigned) and it holds that $C = C()$. If the context is clear, we will sometimes simply speak of circuits when we actually mean circuits with n unassigned inputs for some $n \geq 1$.

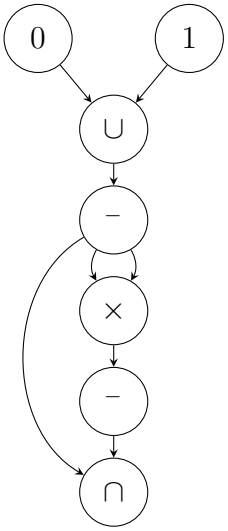
Definition 10.5. Let $\mathcal{O} \subseteq \{^-, \cup, \cap, +, \times\}$. We define *membership* problems, *equivalence* and *satisfiability* problems for circuits and *equivalence problems for formulas* (or expressions).

$$
\begin{aligned}
\mathrm{MC}(\mathcal{O}) &:= \{(C, b) \mid C \text{ is an } \mathcal{O}\text{-circuit and } b \in \mathrm{I}(C)\} \\
\mathrm{EC}(\mathcal{O}) &:= \{(C_1, C_2) \mid C_1, C_2 \text{ are } \mathcal{O}\text{-circuits such that } \mathrm{I}(C_1) = \mathrm{I}(C_2)\} \\
\mathrm{SC}(\mathcal{O}) &:= \{(C, b) \mid C \text{ is an } \mathcal{O}\text{-circuit with } n \text{ unassigned inputs } u_1 < \cdots < u_n, \\
&\qquad\qquad n \geq 0 \text{ and } b \in I\Big(C(x_1, \ldots, x_n)\Big) \text{ for some } x_1, \ldots, x_n \in \mathbb{N}\} \\
\mathrm{EF}(\mathcal{O}) &:= \{(C_1, C_2) \mid C_1, C_2 \text{ are } \mathcal{O}\text{-expressions (formulas) such that } \mathrm{I}(C_1) = \mathrm{I}(C_2)\}
\end{aligned}
$$

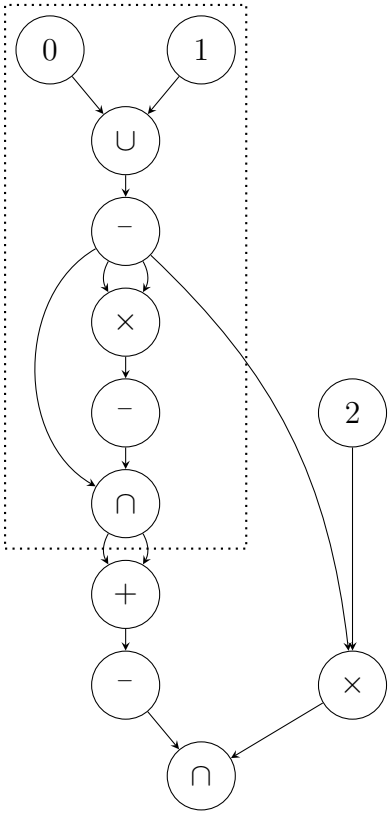
Note that $\mathrm{EF}(\mathcal{O})$ is the problem of testing equivalence for expressions studied by Stockmeyer and Meyer [SM73]. Furthermore, the problem $\mathrm{MC}(\mathcal{O})$ was studied extensively by McKenzie and Wagner [MW07]. Their results are summarized together with our new results in Table 10.1 on page 199.

When an $\mathcal{O}$-circuit $C = (V, E, g_c, \alpha)$ (with or without unassigned inputs) is used as input for an algorithm, all numbers are encoded in binary and furthermore, we use a suitable encoding such that it is possible to verify in deterministic logarithmic space whether a given string encodes a valid circuit. In the following, we will therefore assume that all algorithms start with such a validation of their input strings.

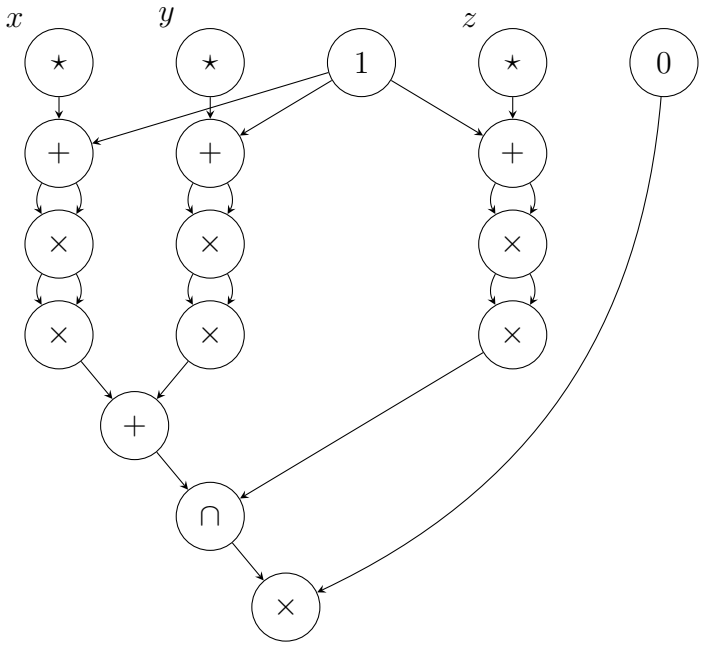
We will notate circuits in two ways: Either graphically as in Figure 10.1 or as expressions over variables, natural numbers and predefined circuits and connect them using the allowed operations. Let us consider the circuits in Figure 10.1 as examples.



(a) Primes



(b) Goldbach's Conjecture



(c) Diophantine Equation

Figure 10.1: Three example circuits.

Example 10.6 ([MW07])**.** Let C_{a} be the the circuit in Figure 10.1(a). It is a more succinct representation of the expression that computes PRIMES. The first $^-$-gate in C_{a} computes the set $\{2, 3, \dots\}$. The following $\times$-gate computes the set of all composite numbers and the last gate computes the set of all primes.

Example 10.7 ([MW07])**.** The circuit C_{a} appears as a subcircuit in the circuit C_{b} shown in Figure 10.1(b). The last $\cap$-gate of C_{b} computes the set $\{p+q \mid p, q \text{ prime}\} - \{x \in \mathbb{N} \mid x > 2 \text{ and } x \text{ even}\}$. Goldbach's conjecture states that every even integer greater than two is the sum of two primes. Note that $\mathrm{I}(C_{\mathrm{b}})$ is non-empty if and only if Goldbach's conjecture holds true. In the introduction we already gave two expressions that are equivalent if and only if Goldbach's conjecture holds. The fact that question can also be expressed as a membership problem is not a coincidence here. We will show that $\mathrm{MC}(^-, \cup, \cap, +, \times)$ and $\mathrm{EC}(^-, \cup, \cap, +, \times)$ are $\leq_{\mathrm{m}}^{\log}$-equivalent (Propositions 10.15 and 10.17.1). If these two problems were decidable, there would be an algorithm that could check if Goldbach's conjecture is true or not. The question whether these two problems are decidable is one of the most intriguing open questions concerning circuits over sets of natural numbers.

Example 10.8. As we will see, undecidability results can be shown for the satisfiability problems for circuits with unassigned inputs. Consider the circuit C_{c} in Figure 10.1(c) and let $a, b, c \in \mathbb{N}$. If we assign the values a, b, c to the unassigned inputs x, y, z of C_{c}, we obtain the circuit $C_{\mathrm{c}}(a, b, c)$. The $\cap$-gate of this circuit computes a non-empty set if and only if $(a+1)^4 + (b+1)^4 = (c+1)^4$. Hence, the circuit computes $\{0\}$ if the equation $x^4 + y^4 = z^4$ has a solution in the positive integers and $\emptyset$ otherwise. Fermat's last theorem thus shows that $(C_{\mathrm{c}}, 0) \notin \mathrm{SC}(\cap, +, \times)$. This circuit exemplifies how Diophantine equation can be modeled using circuits over sets of natural numbers. We will use this technique to show that $\mathrm{SC}(\cap, +, \times)$ is undecidable in Theorem 10.47.

We first note the possibility of some basic constructions that will be used in later proofs.

Lemma 10.9. The following three tasks are solvable in deterministic logarithmic space.

1. On input of a number a output a $\{+\}$-circuit C with inputs from $\{0, 1\}$ such that $\mathrm{I}(C) = \{a\}$.

2. On input of a number a output a $\{+\}$-circuit C with one unassigned input such that for all $x \in \mathbb{N}$, $\mathrm{I}(C(x)) = \{a \cdot x\}$.

3. On input of a multivariate polynomial $p(x_1, \dots, x_n)$ with coefficients from $\mathbb{N}$ output a $\{+, \times\}$-circuit C with n unassigned inputs and all other inputs from $\{0, 1\}$ such that $\mathrm{I}(C(a_1, \dots, a_n)) = \{p(a_1, \dots, a_n)\}$ for all $a_1, \dots, a_n \in \mathbb{N}$.

A fourth task is computable in exponential time for $\mathcal{O} \subseteq \{^-, \cup, \cap, +, \times\}$.

4. Unfold an $\mathcal{O}$-circuit C into an $\mathcal{O}$-expression C' (note that the size of C' can be exponential in the size of C).
   More formally: On input of an $\mathcal{O}$-circuit C (with $n \geq 0$ unassigned inputs) compute an $\mathcal{O}$-circuit C' with n unassigned inputs such that each gate that is not an unassigned input has outdegree at most one and for all $x_1, \dots, x_n \in \mathbb{N}$ it holds that $\mathrm{I}(C(x_1, \dots, x_n)) = \mathrm{I}(C'(x_1, \dots, x_n))$.

Proof. 1. Note that the case $a = 0$ is realized by a single-gate circuit with input 0. Assume now that $a \geq 1$ and let $\mathrm{bin}(a) = a_n \cdots a_0$. We construct a circuit that contains the gates $v_0, \ldots, v_n$ such that v_0 is an input gate with input 1 and all other gates have label $+$. Moreover, for $0 \leq i < n$, there are two edges from gate v_i to gate v_{i+1}. Observe that $\mathrm{I}(v_i) = \{2^i\}$ and therefore,

$$\sum_{\substack{i \in \{0,\ldots,n\} \\ a_i = 1}} \mathrm{I}(v_i) = \{a\}.$$

This sum can be produced by adding at most $n - 1$ additional gates with label $+$ to our circuit and by suitably connecting these new gates to the gates v_i where $a_i = 1$. This construction is possible in logarithmic space and results in a circuit C such that $\mathrm{I}(C) = \{a\}$.

2. Observe that the circuit is obtained simply by changing the input gate v_0 of part 1 to an unassigned input gate. In this case, the gate v_i computes $\{2^i \cdot x\}$ and therefore, the modified circuit computes $\{a \cdot x\}$.

3. We use the first statement to construct all coefficients of the polynomial and then use $+$- and $\times$-gates to suitably connect these constants and the variable inputs.

4. This task is carried out by creating k copies of each gate g and all its direct and indirect predecessors, but not including the variable input gates where g has outdegree $k + 1$. □

10.3 Equivalence Problems

We now want to investigate the complexity of the equivalence problem for various sets of allowed operations. We start by summarizing known results about the complexity of equivalence problems.

Theorem 10.10.

1. [MW07] $\mathrm{EC}(+)$ is $\leq_{\mathrm{m}}^{\log}$-complete for $\mathrm{C_{=}L}$.
2. [SM73] $\mathrm{EF}(\cup, +)$ is $\leq_{\mathrm{m}}^{\log}$-complete for Π_2^{P}.
3. [SM73] $\mathrm{EF}(^{-}, \cup, \cap, +)$ is $\leq_{\mathrm{m}}^{\log}$-complete for PSPACE.
4. [Sch79] $\mathrm{EC}(+, \times)$ is in coRP.

As demonstrated in the introduction, we can generate interesting sets like the set of all primes using only multiplication and the set operations. Contrary to that, circuits whose only arithmetic operation is addition can only compute finite or cofinite sets. The essential difference between the sum and the product in this context is that the sum $A + B$ of two cofinite sets A and B is cofinite whereas $A \times B$ is generally not cofinite; in particular, $A \times B$ excludes all the primes if $1 \notin A \cup B$.

Proposition 10.11. If C is a given circuit over $\mathcal{O} \subseteq \{^{-}, \cup, \cap, +\}$, then there exists an $n \leq 2^{|C|} + 1$ such that for all $z \geq n$,

$$z \in \mathrm{I}(C) \quad \Longleftrightarrow \quad n \in \mathrm{I}(C).$$

Proof. This can be shown by an induction over the size of C. □

Note that by Proposition 10.11, we gain knowledge about the generated set of a $\{^-, \cup, \cap, +\}$-circuit C once we know the membership for all numbers of length at most $|C| + 1$. Clearly, multiplication gates break this property: For instance, the set of all primes is neither finite nor cofinite, but can be generated by a $\{^-, \cup, \cap, \times\}$-circuit.

It is obvious that $\{\cup, \cap, +, \times\}$-circuits can only compute finite sets. However, because of the multiplication gates, we obtain only a double exponential upper bound on the computed numbers.

Proposition 10.12. If C is a $\{\cup, \cap, +, \times\}$-circuit, then $\mathrm{I}(C) \subseteq \{0, 1, \ldots, 2^{2^{2|C|}}\}$.

Proof. This can be shown by an induction over the size of C. □

10.3.1 Relations to Membership Problems

In this subsection we discuss that in some cases (i. e., for several $\mathcal{O} \subseteq \{^-, \cup, \cap, +, \times\}$), the complexity of the equivalence problem $\mathrm{EC}(\mathcal{O})$ is related to the complexity of $\mathrm{MC}(\mathcal{O})$ in a straightforward way. As a consequence, we obtain several general upper and lower bounds for $\mathrm{EC}(\mathcal{O})$ which we summarize below. In contrast, in the following sections more complicated arguments are needed to establish optimal bounds.

Lemma 10.13. If $\mathcal{O} \subseteq \{^-, \cup, \cap, +\}$, then $\mathrm{EC}(\mathcal{O}) \in \mathrm{coNP}^{\mathrm{MC}(\mathcal{O})}$.

Proof. Fix $\mathcal{O} \subseteq \{^-, \cup, \cap, +\}$. By Proposition 10.11, in order to gain complete knowledge about the set generated by an $\mathcal{O}$-circuit C, we only have to look at the first $2^{|C|} + 1$ elements in the set. Thus when comparing the outputs of two circuits C_1 and C_2 over $\mathcal{O}$, it suffices to compare all numbers below and including $\max(2^{|C_1|} + 1, 2^{|C_2|} + 1)$. The circuits generate different sets if and only if one of these comparisons fails. Hence if $\mathrm{I}(C_1) \neq \mathrm{I}(C_2)$ we can guess a witness in polynomial time and query the $\mathrm{MC}(\mathcal{O})$-oracle twice to verify our guess. □

Applying Lemma 10.13 to the results by McKenzie and Wagner [MW07], we obtain:

Corollary 10.14. It holds that

1. $\mathrm{EC}(\cup, +) \in \Pi_2^{\mathrm{P}}$.
2. $\mathrm{EC}(\cup, \cap, +)$, $\mathrm{EC}(^-, \cup, \cap, +) \in \mathrm{PSPACE}$.

The following proposition shows that in many cases, the intuition that equivalence problems are a generalization of membership problems is correct.

Proposition 10.15. If $\{\cap\} \subseteq \mathcal{O}$ or $\{\cup\} \subseteq \mathcal{O}$ or $\mathcal{O} \subseteq \{+, \times\}$, then $\mathrm{MC}(\mathcal{O}) \leq_{\mathrm{m}}^{\log} \mathrm{EC}(\mathcal{O})$.

Proof. Observe that $b \in C \iff C \cup \{b\} = C \iff C \cap \{b\} = \{b\}$. Moreover, circuits over $\{+, \times\}$ only compute singletons. □

In combination with the results by McKenzie and Wagner [MW07] we obtain the following lower bounds.

Corollary 10.16. It holds that

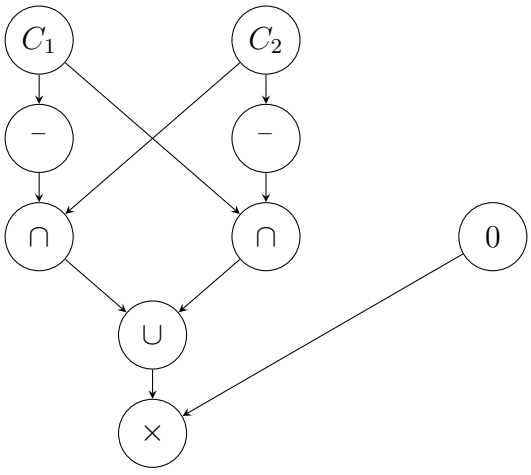
1. $\mathrm{EC}(^{-}, \cup, \cap, +, \times)$ and $\mathrm{EC}(\cup, \cap, +, \times)$ are $\leq_{\mathrm{m}}^{\log}$-hard for NEXP.
2. $\mathrm{EC}(^{-}, \cup, \cap, +)$, $\mathrm{EC}(^{-}, \cup, \cap, \times)$, $\mathrm{EC}(\cup, \cap, +)$, $\mathrm{EC}(\cup, \cap, \times)$ and $\mathrm{EC}(\cup, +, \times)$ are $\leq_{\mathrm{m}}^{\log}$-hard for PSPACE.
3. $\mathrm{EC}(\cup, +)$ and $\mathrm{EC}(\cup, \times)$ are $\leq_{\mathrm{m}}^{\log}$-hard for NP.
4. $\mathrm{EC}(^{-}, \cup, \cap)$, $\mathrm{EC}(\cup, \cap)$, $\mathrm{EC}(\cap, +, \times)$ and $\mathrm{EC}(+, \times)$ are $\leq_{\mathrm{m}}^{\log}$-hard for P.
5. $\mathrm{EC}(\cap, +)$, $\mathrm{EC}(\cap, \times)$ and $\mathrm{EC}(+)$ are $\leq_{\mathrm{m}}^{\log}$-hard for $\mathrm{C}_{=}\mathrm{L}$.
6. $\mathrm{EC}(\cup)$, $\mathrm{EC}(\cap)$ and $\mathrm{EC}(\times)$ are $\leq_{\mathrm{m}}^{\log}$-hard for NL.

It turns out that in some cases, the equivalence problem is not harder than the membership problem.

Proposition 10.17. It holds that

1. If $\{^{-}, \cap, \times\} \subseteq \mathcal{O}$ or $\{^{-}, \cup, \times\} \subseteq \mathcal{O}$, then $\mathrm{EC}(\mathcal{O}) \leq_{\mathrm{m}}^{\log} \mathrm{MC}(\mathcal{O})$.
2. If $\mathcal{O} \subseteq \{^{-}, \cup, \cap\}$, then $\mathrm{EC}(\mathcal{O}) \leq_{\mathrm{T}}^{\log} \mathrm{MC}(\mathcal{O})$.
3. If $\mathcal{O} \subseteq \{+, \times\}$, then $\mathrm{EC}(\mathcal{O}) \leq_{\mathrm{m}}^{\log} \mathrm{MC}(\mathcal{O} \cup \{\cap, \times\})$.
4. If $\mathcal{O} \subseteq \{\cap, +, \times\}$, then $\mathrm{EC}(\mathcal{O}) \leq_{\mathrm{T}}^{\log} \mathrm{MC}(\mathcal{O} \cup \{\cap, \times\})$.

Proof. For statement 1, consider the function $f(C_1, C_2) := (\overline{C}, 0)$ where C denotes the following circuit.



Since $\cup$ can be easily simulated by $\cap$ and $^{-}$ (and similarly for $\cap$), we can assume that C is an $\mathcal{O}$-circuit if C_1 and C_2 are $\mathcal{O}$-circuits for $\{^{-}, \cap, \times\} \subseteq \mathcal{O}$ or $\{^{-}, \cup, \times\} \subseteq \mathcal{O}$.

We now show that f witnesses the reduction $\mathrm{EC}(\mathcal{O}) \leq_{\mathrm{m}}^{\log} \mathrm{MC}(\mathcal{O})$. Note that f is computable in deterministic logarithmic space.

If $(C_1, C_2) \in \mathrm{EC}(\mathcal{O})$, then $\overline{\mathrm{I}(C_1)} \cap \mathrm{I}(C_2) = \emptyset$ and $\mathrm{I}(C_1) \cap \overline{\mathrm{I}(C_2)} = \emptyset$. Hence $\mathrm{I}(C) = \emptyset$. This shows $(\overline{C}, 0) \in \mathrm{MC}(\mathcal{O})$. Otherwise, if $(C_1, C_2) \notin \mathrm{EC}(\mathcal{O})$, then $\mathrm{I}(C_1) \neq \mathrm{I}(C_2)$ and hence, $\overline{\mathrm{I}(C_1)} \cap \mathrm{I}(C_2) \neq \emptyset$ or $\mathrm{I}(C_1) \cap \overline{\mathrm{I}(C_2)} \neq \emptyset$. It follows that $\overline{(\overline{\mathrm{I}(C_1)} \cap \mathrm{I}(C_2))} \cap$

$\overline{(\mathrm{I}(C_1) \cap \overline{\mathrm{I}(C_2)})} \neq \mathbb{N}$ and so $\mathrm{I}(C) = \{0\}$. Therefore, $(\overline{C}, 0) \notin \mathrm{MC}(\mathcal{O})$. This shows $\mathrm{EC}(\mathcal{O}) \leq_{\mathrm{m}}^{\log} \mathrm{MC}(\mathcal{O})$ via reduction function f.

For statement 2, observe that for a $\{^-, \cup, \cap\}$-circuit C with inputs $I \subseteq \mathbb{N}$ and for $x, y \in \mathbb{N} - I$ it holds that

$$x \in C \iff y \in C.$$

Thus for testing the equivalence of two circuits C_1 and C_2, it suffices to verify that the output sets coincide on all inputs and on one additional number.

For statement 3, observe that two $\{+, \times\}$-circuits C_1 and C_2 are equivalent if and only if the circuit $(C_1 \cap C_2) \times 0$ computes $\{0\}$.

For the last statement, let C_1, C_2 be two $\mathcal{O}$-circuits. Now consider the circuits $C_1' := C_1 \times 0, C_2' = C_2 \times 0$ and $C := (C_1 \cap C_2) \times 0$. These are circuits over $\mathcal{O} \cup \{\cap, \times\}$. Since $\mathcal{O}$ contains at most $\cap$ as set operation, C_1 and C_2 produce a single number or the empty set as output.

If C_1 and C_2 are equivalent, then either both circuits produce the empty set or both circuits produce the same value. In the first case, $(C_1', 0) \notin \mathrm{MC}(\mathcal{O} \cup \{\cap, \times\}), (C_2', 0) \notin \mathrm{MC}(\mathcal{O} \cup \{\cap, \times\}), (C, 0) \notin \mathrm{MC}(\mathcal{O} \cup \{\cap, \times\})$ and in the second case, $(C_1', 0) \in \mathrm{MC}(\mathcal{O} \cup \{\cap, \times\}), (C_2', 0) \in \mathrm{MC}(\mathcal{O} \cup \{\cap, \times\})$ and $(C, 0) \in \mathrm{MC}(\mathcal{O} \cup \{\cap, \times\})$.

If C_1 and C_2 are not equivalent, then either one of them produces the empty set (without loss of generality let C_1 be that circuit) while the other one produces a number or both produce numbers but different ones. In the first case, $(C_1', 0) \notin \mathrm{MC}(\mathcal{O} \cup \{\cap, \times\}), (C_2', 0) \in \mathrm{MC}(\mathcal{O} \cup \{\cap, \times\}), (C, 0) \notin \mathrm{MC}(\mathcal{O} \cup \{\cap, \times\})$ and in the second case, $(C_1', 0) \in \mathrm{MC}(\mathcal{O} \cup \{\cap, \times\}), (C_2', 0) \in \mathrm{MC}(\mathcal{O} \cup \{\cap, \times\}), (C, 0) \notin \mathrm{MC}(\mathcal{O} \cup \{\cap, \times\})$.
So we can test for equivalence just by verifying that the tests $(C_1', 0) \in \mathrm{MC}(\mathcal{O} \cup \{\cap, \times\})$, $(C_2', 0) \in \mathrm{MC}(\mathcal{O} \cup \{\cap, \times\})$, and $(C, 0) \in \mathrm{MC}(\mathcal{O} \cup \{\cap, \times\})$ yield the same result. □

Corollary 10.18. $\mathrm{EC}(\cup, \cap)$, $\mathrm{EC}(^-, \cup, \cap)$, $\mathrm{EC}(\cap, \times)$ and $\mathrm{EC}(\times) \in \mathrm{P}$,
$\mathrm{EC}(\cup)$ and $\mathrm{EC}(\cap) \in \mathrm{NL}$.

Proof. This follows from Proposition 10.17 together with the results by McKenzie and Wagner [MW07]: For $\mathrm{EC}(\cup, \cap)$ and $\mathrm{EC}(^-, \cup, \cap)$ we use Proposition 10.17.2 and the result that $\mathrm{MC}(^-, \cup, \cap) \in \mathrm{P}$. For $\mathrm{EC}(\cap, \times)$ and $\mathrm{EC}(\times)$, Proposition 10.17.4 together with $\mathrm{MC}(\cap, \times) \in \mathrm{P}$ yields the assertion. The second part follows from Proposition 10.17.2 and $\mathrm{MC}(\cup)$, $\mathrm{MC}(\cap) \in \mathrm{NL}$ since NL is closed under $\leq_{\mathrm{T}}^{\log}$ by the theorem of Immerman and Szelepcsényi [Imm88, Sze88]. □

10.3.2 Feasible Equivalence Problems

In this section, we present several equivalence problems for which we can show that efficient algorithms exist. While most of the algorithms require deterministic polynomial time or less, randomness is needed for $\mathrm{EC}(\cap, +, \times)$.

We show that $\mathrm{EC}(\cap, +)$ is complete for the class $\mathrm{coC_{=}L}(2)$, the complement of the second level of the Boolean hierarchy over $\mathrm{C_{=}L}$ (cf. Definitions 2.7 and 2.10). As a useful tool, we introduce non-emptiness problems for circuits.

Definition 10.19. Let $\mathcal{O} \subseteq \{^-, \cup, \cap, +, \times\}$. The *non-emptiness problem* for $\mathcal{O}$-circuits is defined as follows:

$$\mathrm{NEC}(\mathcal{O}) := \{C \mid C \text{ is an } \mathcal{O}\text{-circuit such that } \mathrm{I}(C) \neq \emptyset\}$$

Lemma 10.20. NEC$(\cap, +)$ is $\leq_{\mathrm{m}}^{\log}$-complete for C$_=$L.

Proof. Locally in this proof, we use an additional gate type, the id-gate. Such a gate has indegree one and computes the identity. Observe that an $\{\mathrm{id}, +\}$-circuit C can be translated in logarithmic space into a $\{+\}$-circuit C' such that $\mathrm{I}(C) = \mathrm{I}(C')$. For this we only have to replace all id-gates by +-gates whose second input is connected to a new input gate with label 0. Hence EC$(\mathrm{id}, +) \leq_{\mathrm{m}}^{\log}$ EC$(+)$. So by Theorem 10.10.1, EC$(\mathrm{id}, +) \in$ C$_=$L.

We define the function f: For a given $\{\cap, +\}$-circuit C and a given gate number k it outputs a new $\{\mathrm{id}, +\}$-circuit by deleting the second input edge of each $\cap$-gate and turning it into an id-gate and changing the output gate to the gate with number k. Observe that f is computable in logarithmic space.

Claim 10.21. Let C be an $\{\cap, +\}$-circuit. If g is a gate in C such that $X := \mathrm{I}_C(g)$ is non-empty, then $\mathrm{I}_{f(C,k)}(g) = X$ for any gate number k.

Proof of the claim. Assume the claim does not hold. Choose the first (smallest) gate g in C and some k such that $X := \mathrm{I}_C(g)$ is non-empty and $\mathrm{I}_{f(C,k)}(g) \neq X$. It is obvious that g cannot be an input gate. From the choice of g it follows that any direct predecessor of g in C computes the same set in C and in $f(C, k)$. Since an $\{\cap, +\}$-circuit can only compute sets with cardinality at most 1 and $X \neq \emptyset$, all direct predecessors of g must compute singletons. If g has label +, then it is not modified by f and thus must compute the same set in $f(C, k)$. If g has label $\cap$, then all direct predecessors of g compute the same singleton and thus, g still computes this singleton if g's second input edge is deleted by f. This contradicts our assumption and proves Claim 10.21. ◇

Consider the function h given by Algorithm 10.1 that on input of an $\{\cap, +\}$-circuit C generates a list of pairs of $\{\mathrm{id}, +\}$-circuits $(C_{1,1}, C_{1,2})$, $(C_{2,1}, C_{2,2})$, …, $(C_{r,1}, C_{r,2})$. Later we will show that h is a conjunctive truth-table reduction from NEC$(\cap, +)$ to EC$(\mathrm{id}, +)$.

Algorithm 10.1: `NEC_reduction(`C`)`, $h(C)$

```
   Input : {∩,+}-circuit C = (V, E, g_C, α) with n gates
   Output: List of pairs of {id,+}-circuits
1  for i := 1 to n do
2      if gate i has label ∩ and is a direct or indirect predecessor of g_C or i = g_C
       then
3          let g_1 and g_2 be the direct predecessors of gate i;
4          let C_1 = f(C, g_1) and C_2 = f(C, g_2);
5          output (C_1, C_2);
```

Observe that for circuits, the test whether gate i is connected to g_C (line 2) can be carried out by one query to an NL-oracle. Hence h is computable in deterministic logarithmic space with the help of an NL-oracle. Thus we have $h \in$ FNL, where FNL $=$ FL$^{\mathrm{NL}}$ denotes the class of functions computable in deterministic logarithmic space with access to an NL-oracle [ÀBJ95]. Note that FNL $=$ NLSV$_{\mathrm{t}}$ (cf. Definition 3.6).

Claim 10.22. $\mathrm{NEC}(\cap,+)$ conjunctively truth-table reduces to $\mathrm{EC}(\mathrm{id},+)$ via the function h. Formally, for every $\{\cap,+\}$-circuit C, if $h(C) = (C_{1,1}, C_{1,2})$, $(C_{2,1}, C_{2,2})$,..., $(C_{r,1}, C_{r,2})$, then

$$C \in \mathrm{NEC}(\cap,+) \quad \Longleftrightarrow \quad \bigwedge_{j=1}^{r} (C_{j,1}, C_{j,2}) \in \mathrm{EC}(\mathrm{id},+). \tag{10.1}$$

Proof of the claim. "$\Leftarrow$": Assume $C \notin \mathrm{NEC}(\cap,+)$, i. e., $\mathrm{I}(C) = \emptyset$. Let g be the smallest gate that is connected to the output gate and computes the empty set. For the direct predecessors g_1 and g_2 of g it holds that $\emptyset \neq \mathrm{I}(g_1) \neq \mathrm{I}(g_2) \neq \emptyset$. By Claim 10.21, for all k, it holds that $\mathrm{I}_C(g_1) = \mathrm{I}_{f(C,k)}(g_1)$ and $\mathrm{I}_C(g_2) = \mathrm{I}_{f(C,k)}(g_2)$. So for all k, the gates $\mathrm{I}_{f(C,k)}(g_1) \neq \mathrm{I}_{f(C,k)}(g_2)$. Therefore, $C_1 := f(C, g_1)$ and $C_2 := f(C, g_2)$ compute different sets and hence $(C_1, C_2) \notin \mathrm{EC}(\mathrm{id},+)$. The pair (C_1, C_2) appears on the list $h(C)$, since g is connected to the output gate g_C and has label $\cap$. Therefore, the right-hand side of (10.1) is false.

"$\Rightarrow$": Assume $C \in \mathrm{NEC}(\cap,+)$, i. e., $\mathrm{I}(C) \neq \emptyset$. Fix any $1 \leq j \leq r$. The pair $(C_{j,1}, C_{j,2})$ appears on the list $h(C)$, because at a certain time, the algorithm made the output $(C_{j,1}, C_{j,2})$ in line 5. Assume that at this time the variable i had the value k. So the gate with number k is connected to the output gate g_C and has the label $\cap$. Let g_1 and g_2 be the direct predecessors of the gate with number k. So $C_{j,1} = f(C, g_1)$ and $C_{j,2} = f(C, g_2)$. The gates k, g_1, and g_2 are connected to g_C and therefore none of them computes the empty set. It follows that all three gates must compute the same singleton. So by Claim 10.21, $f(C, g_1)$ and $f(C, g_2)$ compute the same singleton. Therefore, $(C_{j,1}, C_{j,2}) = (f(C, g_1), f(C, g_2)) \in \mathrm{EC}(\mathrm{id},+)$. This shows that the right-hand side of (10.1) is true. ◇

By Claim 10.22, $\mathrm{NEC}(\cap,+)$ conjunctively truth-table reduces to $\mathrm{EC}(\mathrm{id},+)$ via a function from FNL, i. e., $\mathrm{NEC}(\cap,+) \leq_{\mathrm{ctt}}^{\mathrm{FNL}} \mathrm{EC}(\mathrm{id},+)$. We have seen that $\mathrm{EC}(\mathrm{id},+) \in \mathrm{C_{=}L}$. Allender and Ogihara [AO96] show that $\mathrm{C_{=}L}$ is closed under $\leq_{\mathrm{ctt}}^{\mathrm{FNL}}$. Therefore, $\mathrm{NEC}(\cap,+) \in \mathrm{C_{=}L}$.

McKenzie and Wagner [MW07] show that $\mathrm{MC}(+)$ is $\leq_{\mathrm{m}}^{\log}$-complete for $\mathrm{C_{=}L}$. Observe that for an arbitrary $\{+\}$-circuit C and a number k it holds that $(C, k) \in \mathrm{MC}(+)$ if and only if the circuit $C \cap k \in \mathrm{NEC}(\cap,+)$. So $\mathrm{MC}(+) \leq_{\mathrm{m}}^{\log} \mathrm{NEC}(\cap,+)$. This shows that $\mathrm{NEC}(\cap,+)$ is $\leq_{\mathrm{m}}^{\log}$-complete for $\mathrm{C_{=}L}$. □

The next proposition shows that in some cases, equivalence problems with addition can very easily be reduced to equivalence problems with multiplication.

Proposition 10.23. $\mathrm{EC}(\cap,+) \leq_{\mathrm{m}}^{\log} \mathrm{EC}(\cap,\times)$ and $\mathrm{EC}(+) \leq_{\mathrm{m}}^{\log} \mathrm{EC}(\times)$.

Proof. Because of Lemma 10.9.1, we can assume that a given $\{\cap,+\}$- or $\{+\}$-circuit C only has inputs from $\{0,1\}$ by replacing the inputs greater than 1 by $\{+\}$-circuits with inputs from $\{0,1\}$.

Let g be the function that translates a given $\{\cap,+\}$-circuit C with inputs from $\{0,1\}$ into the following $\{\cap,\times\}$-circuit $g(C)$: All $+$-gates become $\times$-gates, all inputs 0 become 1, and all inputs 1 become 2. Note that g is computable in deterministic logarithmic space and this is why we first transformed all constants into circuits with $\{0,1\}$-inputs. Observe that $\mathrm{I}(C) = \{x\}$ if and only if $\mathrm{I}(g(C)) = \{2^x\}$. Moreover, if C is a $\{+\}$-circuit, then $g(C)$ is a $\{+\}$-circuit.

Now it is easy to see that the function $(C_1, C_2) \mapsto (g(f(C_1)), g(f(C_2)))$ performs both reductions, $\mathrm{EC}(+) \leq_{\mathrm{m}}^{\log} \mathrm{EC}(\times)$ and $\mathrm{EC}(\cap, +) \leq_{\mathrm{m}}^{\log} \mathrm{EC}(\cap, \times)$. □

Theorem 10.24. $\mathrm{EC}(\cap, +)$ is $\leq_{\mathrm{m}}^{\log}$-complete for $\mathrm{coC_{=}L}(2)$.

Proof. Recall that $\mathrm{C_{=}L}(2) = \{A \cap \overline{B} \mid A, B \in \mathrm{C_{=}L}\}$ and thus

$$\begin{aligned}\mathrm{coC_{=}L}(2) &= \{\overline{A \cap \overline{B}} \mid A, B \in \mathrm{C_{=}L}\} \\ &= \{\overline{A} \cup B \mid A, B \in \mathrm{C_{=}L}\} \\ &= \mathrm{C_{=}L} \vee \mathrm{coC_{=}L}.\end{aligned}$$

We start the proof by showing that $\mathrm{EC}(\cap, +)$ belongs to $\mathrm{C_{=}L} \vee \mathrm{coC_{=}L}$.

$$\begin{aligned}A_1 &:= \{(C_1, C_2) \mid C_1, C_2 \text{ are } \{\cap, +\}\text{-circuits and } C_1 \cap C_2 \in \mathrm{NEC}(\cap, +)\} \\ A_2 &:= \{(C_1, C_2) \mid C_1, C_2 \text{ are } \{\cap, +\}\text{-circuits and } C_1, C_2 \notin \mathrm{NEC}(\cap, +)\}\end{aligned}$$

By Lemma 10.20, $A_1 \in \mathrm{C_{=}L}$. From Lemma 10.20 and the fact that $\mathrm{C_{=}L}$ is closed under union [AO96] it follows that $A_2 \in \mathrm{coC_{=}L}$. Observe that $\mathrm{EC}(\cap, +) = A_1 \cup A_2$. Therefore, $\mathrm{EC}(\cap, +) \in \mathrm{C_{=}L} \vee \mathrm{coC_{=}L}$.

Now we show that $\mathrm{EC}(\cap, +)$ is $\leq_{\mathrm{m}}^{\log}$-hard for $\mathrm{C_{=}L} \vee \mathrm{coC_{=}L}$. Let

$$\begin{aligned}B := \{(C_1, k_1, C_2, k_2) \mid &C_1, C_2 \text{ are } \{+\}\text{-circuits and} \\ &((C_1, k_1) \in \mathrm{MC}(+) \text{ or } (C_2, k_2) \notin \mathrm{MC}(+))\}\end{aligned}$$

The set B is $\leq_{\mathrm{m}}^{\log}$-complete for $\mathrm{C_{=}L} \vee \mathrm{coC_{=}L}$, since $\mathrm{MC}(+)$ is $\leq_{\mathrm{m}}^{\log}$-complete for $\mathrm{C_{=}L}$ [MW07]. We show $B \leq_{\mathrm{m}}^{\log} \mathrm{EC}(\cap, +)$ via the following reduction function f.

$$f(C_1, k_1, C_2, k_2) := (C, C'), \quad \text{where } C = (C_2 \cap k_2) + k_1 \text{ and } C' = (C_2 \cap k_2) + C_1.$$

Assume $(C_1, k_1, C_2, k_2) \in B$. If $(C_2, k_2) \notin \mathrm{MC}(+)$, then $\mathrm{I}(C) = \mathrm{I}(C') = \emptyset$ and hence $(C, C') \in \mathrm{EC}(\cap, +)$. So assume now $(C_1, k_1) \in \mathrm{MC}(+)$. If $\mathrm{I}(C) \neq \emptyset$, then $(C, C') \in \mathrm{EC}(\cap, +)$. Otherwise, $\mathrm{I}(C) = \emptyset$ and hence, $\mathrm{I}(C') = \emptyset$ and $(C, C') \in \mathrm{EC}(\cap, +)$.

Assume $(C_1, k_1, C_2, k_2) \notin B$. So $(C_1, k_1) \notin \mathrm{MC}(+)$ and $(C_2, k_2) \in \mathrm{MC}(+)$. It follows that $\mathrm{I}(C) = \{k_2 + k_1\}$ and $\mathrm{I}(C') = \{k_2\} + \mathrm{I}(C_1)$ where $k_1 \notin \mathrm{I}(C_1)$. Therefore, $(C, C') \notin \mathrm{EC}(\cap, +)$. This shows $B \leq_{\mathrm{m}}^{\log} \mathrm{EC}(\cap, +)$ via f. □

Corollary 10.25. $\mathrm{EC}(\cap, \times)$ is $\leq_{\mathrm{m}}^{\log}$-hard for $\mathrm{coC_{=}L}(2)$, $\mathrm{EC}(\times)$ is $\leq_{\mathrm{m}}^{\log}$-hard for $\mathrm{C_{=}L}$.

Proof. Follows from Proposition 10.23, Theorem 10.24, and the fact that $\mathrm{EC}(+)$ is $\leq_{\mathrm{m}}^{\log}$-complete for $\mathrm{C_{=}L}$ [MW07]. □

By Theorem 10.10.4 $\mathrm{EC}(+, \times) \in \mathrm{coRP}$, and McKenzie and Wagner showed [MW07] that $\mathrm{MC}(\cap, +, \times) \equiv_{\mathrm{m}}^{\log} \mathrm{EC}(+, \times)$. So $\mathrm{MC}(\cap, +, \times) \in \mathrm{coRP}$. By Proposition 10.17.4, $\mathrm{EC}(\cap, +, \times) \in \mathrm{P}^{\mathrm{coRP}} \subseteq \mathrm{BPP}^{\mathrm{BPP}} = \mathrm{BPP}$ due to the self-lowness of BPP (Theorem 2.15, [Ko82]).

Corollary 10.26. $\mathrm{EC}(\cap, +, \times) \in \mathrm{BPP}$.

10.3.3 Π_2^{P}-Complete Problems

We prove Π_2^{P}-completeness for $\mathrm{EC}(\cup,+)$ and $\mathrm{EC}(\cup,\times)$. Note that Stockmeyer and Meyer [SM73] showed Π_2^{P}-completeness for equivalence of *expressions* over $\{\cup,+\}$. It was shown by Glaßer et al. [GHR+10] that this problem is still Π_2^{P}-complete for expressions over $\{\cup,\cap,+,\times\}$ (note that this section is based on that article but does not include the results about expressions). As we will see in section 10.3.4, the problem for $\{\cup,\cap,+,\times\}$-circuits gets substantially harder.

We first concentrate on $\mathrm{EC}(\cup,+)$. Here, Corollary 10.14 already shows that $\mathrm{EC}(\cup,+)$ is in Π_2^{P}, hence it suffices to prove Π_2^{P}-hardness for $\mathrm{EC}(\cup,+)$. Useful problems here are quantified Boolean formulas and quantified products of subsets (similar to the sum of subsets problem).

Definition 10.27. We define the following problems.

$$\begin{aligned}
\mathrm{QBF} &:= \{H \mid H \text{ is a Boolean formula in 3-CNF with variables } x_1,\dots,x_{2n} \\
&\qquad \text{such that } \underbrace{\forall x_1 \exists x_2 \cdots \exists x_{2n}}_{\text{strict alternation}} \colon H(x_1,\dots,x_{2n}) = 1\} \\
\mathrm{QBF}_2 &:= \Big\{H \mid H \text{ is a Boolean formula in 3-CNF with variables } x_1,\dots,x_{2n} \\
&\qquad \text{such that } \forall x_1 \forall x_2 \dots \forall x_n \exists x_{n+1} \exists x_{n+2} \dots \exists x_{2n} \colon H(x_1,\dots,x_{2n}) = 1)\} \\
\mathrm{QPOS}_2 &:= \Big\{(x_1,\dots,x_{2n},b) \mid x_1,\dots,x_{2n},b \geq 1 \text{ and } \forall I \subseteq \{1,\dots,n\} \\
&\qquad \exists J \subseteq \{n+1,\dots,2n\} \colon \textstyle\prod_{i\in I} x_i \prod_{j\in J} x_j = b\Big\}
\end{aligned}$$

Stockmeyer and Meyer [SM73] showed that QBF is PSPACE-complete and there are restrictions of this problem that are complete for each level in the polynomial-time hierarchy. For instance, the problem QBF_2 is complete for Π_2^{P} [SM73]. We want to show that its translation to quantified products of subsets, QPOS_2 is also Π_2^{P}-complete.

Lemma 10.28. $\mathrm{QBF}_2 \leq_{\mathrm{m}}^{\log} \mathrm{QPOS}_2$.

Proof. We define a logspace computable function f such that $H \in \mathrm{QBF}_2 \Leftrightarrow f(H) \in \mathrm{QPOS}_2$. Let $H = \bigwedge_{i=1}^{m}(z_{i_1} \vee z_{i_2} \vee z_{i_3})$ with $z_{i_j} \in \{x_1,\dots,x_{2n},\overline{x_1},\dots,\overline{x_{2n}}\}$ be a Boolean formula. We then define f as

$$f(H) := \Big(v_1,\dots,v_n,\underbrace{1,\dots,1}_{2n+2m},v_{n+1},\dots,v_{2n},v'_1,\dots,v'_{2n},p_1,\dots,p_m,p_1^2,\dots,p_m^2,b\Big),$$

where p_i is the i-th prime and $v_1,\dots,v_{2n},v'_1,\dots,v'_{2n}$ and b are the following natural numbers:

$$\begin{aligned}
v_i &:= p_{m+i} \prod_{r=1}^{m} p_r^{k_{r,i}}, \text{ where} \\
&\qquad k_{r,i} \text{ is the number of occurrences of the literal } x_i \text{ in the } r\text{-th clause of } H, \\
v'_i &:= p_{m+i} \prod_{r=1}^{m} p_r^{k'_{r,i}}, \text{ where} \\
&\qquad k'_{r,i} \text{ is the number of occurrences of the literal } \overline{x_i} \text{ in the } r\text{-th clause of } H, \\
b &:= \prod_{i=1}^{2n} p_{m+i} \prod_{r=1}^{m} p_r^4.
\end{aligned}$$

By defining f in this way, we achieve that the numbers can actually be interpreted as vectors consisting of the exponents in their prime number decomposition and all numbers are appropriately quantified in the QPOS_2-instance: $v_1, \ldots, v_n$ are all quantified universally and all other numbers (except for b, of course) are quantified existentially.

In the following, let $I_{a_1,\ldots,a_{2n}}$ be the interpretation that, for $1 \leq i \leq 2n$, assigns the truth-value $a_i \in \{0,1\}$ to the variable x_i in H. Also, let $\overline{I}$ for $I \subseteq \{1,\ldots,n\}$ denote $\{1,\ldots,n\} - I$ and $\overline{J} := \{n+1,\ldots,2n\} - J$ for $J \subseteq \{n+1,\ldots,2n\}$.

The following equivalences now hold:

$$
\begin{aligned}
H \in \mathrm{QBF}_2 &\Leftrightarrow \forall a_1 \cdots \forall a_n \exists a_{n+1} \cdots \exists a_{2n} (I_{a_1,\ldots,a_{2n}} \text{ satisfies } H) \\
&\Leftrightarrow \forall a_1 \cdots \forall a_n \exists a_{n+1} \cdots \exists a_{2n} (I_{a_1,\ldots,a_{2n}} \text{ satisfies each clause of } H) \\
&\overset{(\star)}{\Leftrightarrow} \forall_{I \subseteq \{1,\ldots,n\}} \exists_{J \subseteq \{n+1,\ldots,2n\}} \exists_{k_1,\ldots,k_m \in \{1,2,3\}} \text{ such that} \\
&\qquad \prod_{i \in I} v_i \prod_{i \in \overline{I}} v'_i \prod_{j \in J} v_j \prod_{j \in \overline{J}} v'_j = \prod_{i=1}^{2n} p_{m+i} \prod_{r=1}^{m} p_r^{k_r} \\
&\Leftrightarrow \forall_{I \subseteq \{1,\ldots,n\}} \exists_{J \subseteq \{n+1,\ldots,2n\}} \exists_{R_1 \subseteq \{1,\ldots,m\}} \exists_{R_2 \subseteq \{1,\ldots,m\}} \text{ such that} \\
&\qquad \prod_{i \in I} v_i \prod_{i \in \overline{I}} v'_i \prod_{j \in J} v_j \prod_{j \in \overline{J}} v'_j \prod_{r \in R_1} p_r \prod_{r \in R_2} p_r^2 = \prod_{i=1}^{2n} p_{m+i} \prod_{r=1}^{m} p_r^4 = b \\
&\Leftrightarrow \forall_{I \subseteq \{1,\ldots,n\}} \exists_{J \subseteq \{n+1,\ldots,2n\}} \exists_{I' \subseteq \{1,\ldots,2n\}} \exists_{R_1 \subseteq \{1,\ldots,m\}} \exists_{R_2 \subseteq \{1,\ldots,m\}} \text{ such that} \\
&\qquad \prod_{i \in I} v_i \prod_{j \in J} v_j \prod_{i \in I'} v'_i \prod_{r \in R_1} p_r \prod_{r \in R_2} p_r^2 = b \\
&\Leftrightarrow f(H) \in \mathrm{QPOS}_2.
\end{aligned}
$$

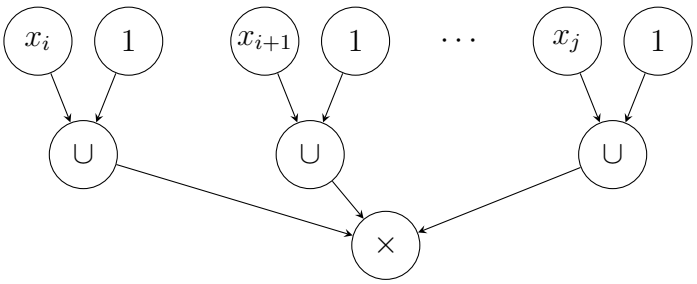
To see $(\star)$, observe that the first product on the right hand side enforces that for each variable exactly one value out of $\{0,1\}$ is chosen (this is actually only needed later on), and the second product makes sure that in each clause at least one literal is true (the exponents k_j are not zero).

To see that f is computable in logarithmic space it remains to show that the primes $p_1, \ldots, p_m$ can be generated in space logarithmic in m. One can show that $p_m \leq O(m^2)$ using the prime number theorem. Since the length of $O(m^2)$ is logarithmic in m, we can obtain $p_1, \ldots, p_m$ by brute force primality tests of $1, \ldots, O(m^2)$ in space logarithmic in m. This completes the reduction. □

Corollary 10.29. QPOS_2 is $\leq_{\mathrm{m}}^{\log}$-complete for Π_2^{P}.

Theorem 10.30. $\mathrm{EC}(\cup, \times)$ is $\leq_{\mathrm{m}}^{\log}$-hard for Π_2^{P}.

Proof. We describe a reduction from QPOS_2 to $\mathrm{EC}(\cup, \times)$. To model the products in the definition of QPOS_2 we define the circuits $C_{i,j}$ for a QPOS_2-instance $(x_1, \ldots, x_{2n}, b)$ and $1 \leq i \leq j \leq 2n$ as shown in the following diagram.



Note that the final product gate has to be expanded appropriately. We obviously have $\mathrm{I}(C_{i,j}) = \{\prod_{k\in I} x_k \mid I \subseteq \{i, i+1, \ldots, j\}\}$. Additionally, let $C_y := \prod_{j=n+1}^{2n} x_j$.

Our reduction function f is now defined as follows: If the input to the function is not a tuple, has an even number of elements, or one of the elements is zero, then the function returns $(0, 1)$. Otherwise we define:

$$f((x_1, \ldots, x_{2n}, b)) := ((C_{1,n} \times C_y) \cup (b \times C_{n+1,2n}), (b \times C_{n+1,2n}))$$

The correctness is observed as follows.

$$\begin{aligned}
&(x_1, \ldots, x_n, x_{n+1}, \ldots, x_{2n}, b) \in \mathrm{QPOS}_2 \\
&\quad\iff \forall_{I\subseteq\{1,\ldots,n\}}\exists_{J\subseteq\{n+1,\ldots,2n\}} \text{ such that } \prod_{i\in I} x_i \prod_{j\in J} x_j = b \\
&\quad\iff \forall_{I\subseteq\{1,\ldots,n\}}\exists_{J\subseteq\{n+1,\ldots,2n\}} \text{ such that } \prod_{i\in I} x_i \prod_{j=n+1}^{2n} x_j = b \prod_{j\notin J} x_j \\
&\quad\iff \forall_{I\subseteq\{1,\ldots,n\}}\exists_{J\subseteq\{n+1,\ldots,2n\}} \text{ such that } \prod_{i\in I} x_i \prod_{j=n+1}^{2n} x_j = b \prod_{j\in J} x_j \\
&\quad\iff \forall l \in \mathrm{I}(C_{1,n} \times C_y)\exists r \in \mathrm{I}(b \times C_{n+1,2n}) \text{ such that } l = r \\
&\quad\iff \mathrm{I}(C_{1,n} \times C_y) \subseteq \mathrm{I}(b \times C_{n+1,2n}) \\
&\quad\iff \mathrm{I}((C_{1,n} \times C_y) \cup (b \times C_{n+1,2n})) = \mathrm{I}(b \times C_{n+1,2n}) \\
&\quad\iff ((C_{1,n} \times C_y) \cup (b \times C_{n+1,2n}), (b \times C_{n+1,2n})) \in \mathrm{EC}(\cup, \times)
\end{aligned}$$

Again, f can be computed in logarithmic space. Thus, by Corollary 10.29, $\mathrm{EC}(\cup, \times)$ is $\leq_{\mathrm{m}}^{\log}$-hard for Π_2^{P}. □

In order to show that $\mathrm{EC}(\cup, \times)$ belongs to Π_2^{P} we reduce $\mathrm{EC}(\cup, \times)$ to $\mathrm{EC}(\cup, +)$. The idea is as follows: We represent input numbers of the $\{\cup, \times\}$-circuits as products of the form $q_1^{e_1} q_2^{e_2} \cdots q_m^{e_m}$ and we build corresponding $\{\cup, +\}$-circuits that generate exactly the vectors $(e_1, e_2, \ldots, e_m)$ which we will appropriately encode as numbers. Note that a factorization into prime factors q_i would be welcome, but it is unknown if this is possible in polynomial time. However, as demonstrated by McKenzie and Wagner, for our purpose, a factorization into factors that are relatively prime suffices. To this end we need the following problem.

Definition 10.31 ([BS96])**.** The computation of a gcd-free basis is the problem that can be formalized as the following multivalued function:

$$\begin{aligned}
\mathrm{GFB}(\langle n, \langle a_1, \ldots, a_n\rangle\rangle) = \{\langle Q, E\rangle \mid n \geq 1, \text{ and } (Q, E) \text{ is a gcd-free basis} \\
\text{for } a_1, \ldots, a_n \geq 1\}
\end{aligned}$$

Where (Q, E) is called a *gcd-free basis for the numbers* $a_1, \ldots, a_n$ if $Q = \langle\{q_1, \ldots, q_m\}\rangle$, $m \geq 1$, $q_i \geq 2$, q_i and q_j are relatively prime for $i \neq j$, $E = (e_{ij}) \in \mathbb{N}^{n\times m}$ and $a_i = \prod_{j=1}^{m} q_j^{e_{ij}}$ for all $1 \leq i \leq n$.

So on input of some target numbers $a_1, \ldots, a_n$ the task is to compute a basis of relatively prime numbers $q_1, \ldots, q_n$ and exponents e_{ij} for each combination of basis and target number such that $a_i = \prod_{j=1}^{m} q_j^{e_{ij}}$ for all $1 \leq i \leq n$.

Proposition 10.32 ([BS96])**.** A gcd-free basis can be computed in polynomial time in the sense that the multivalued function GFB is polynomial-time solvable.

We start by establishing the reduction for $\{\cup, \times\}$-circuits with positive inputs.

Lemma 10.33. There exists a polynomial-time computable function f that maps pairs of $\{\cup, \times\}$-circuits to pairs of $\{\cup, +\}$-circuits such that for any two $\{\cup, \times\}$-circuits C_1 and C_2 with positive inputs,

$$(C_1, C_2) \in \mathrm{EC}(\cup, \times) \quad \Longleftrightarrow \quad f(C_1, C_2) \in \mathrm{EC}(\cup, +).$$

Proof. We transform the $\{\cup, \times\}$-circuit into a $\{\cup, +\}$-circuit with the same structure such that the transformed circuit operates on the exponents of a gcd-free basis of the inputs. To this end, we replace each $\times$-node by a $+$-node and each input by a number whose binary representation consists of blocks of fixed length. Each such block contains the exponent of one component of the base. Also, the blocks are long enough to ensure that they do not interfere with each other via carries.

Let C_1 and C_2 be two $\{\cup, \times\}$-circuits. If their numbers of input gates do not match, introduce additional input gates with assignment one that are multiplied to the output gate. We denote the inputs of C_1 and C_2 by $x_1, \ldots, x_n$ and $\tilde{x}_1, \ldots, \tilde{x}_n$, respectively. Now compute a gcd-free basis $q_1, \ldots, q_m$ for $x_1, \ldots, x_n, \tilde{x}_1, \ldots, \tilde{x}_n$. By Proposition 10.32, this is possible in polynomial time (note that all inputs are positive). Let $Q := \{\prod_{j=1}^{m} q_j^{e_j} \mid e_j \in \mathbb{N}\}$ be the multiplicative monoid generated by $q_1, \ldots, q_m$. For $x = \prod_{j=1}^{m} q_j^{e_j} \in Q$, let $\varepsilon(x, j) = e_j$ be the unique j-th exponent in the representation of x.

For $N := 2^{2|C_1|+2|C_2|}$ we now define a monoid homomorphism $\sigma\colon (Q, \times) \to (\mathbb{N}, +)$ by $x \mapsto \sum_{j=1}^{m} \varepsilon(x, j) N^{j-1}$. Note that σ is injective on $X := \{x \in Q \mid \varepsilon(x, j) < N \text{ for all } 1 \leq j \leq m\}$.

Since all sets generated by gates of C_1 and C_2 are subsets of Q, we can define the function f by $f(C_1, C_2) = (C_1', C_2')$ where C_1' results from C_1 by replacing all $\times$-gates by $+$-gates and replacing the inputs x_i by $\sigma(x_i)$; C_2' is obtained analogously. Since σ is a homomorphism, we obtain $\sigma(\mathrm{I}(C_1)) = \mathrm{I}(C_1')$ and $\sigma(\mathrm{I}(C_2)) = \mathrm{I}(C_2')$.

By Proposition 10.12, for all numbers x produced by gates of the circuits C_1 and C_2 it holds that $x < 2^{2^{2|C_1|+2|C_2|}}$, and thus $\varepsilon(x, j) < 2^{2|C_1|+2|C_2|} = N$. Therefore, σ is injective in $\mathrm{I}(C_1)$ and $\mathrm{I}(C_2)$ and we obtain $\mathrm{I}(C_1) = \mathrm{I}(C_2) \iff \sigma(\mathrm{I}(C_1)) = \sigma(\mathrm{I}(C_2)) \iff \mathrm{I}(C_1') = \mathrm{I}(C_2')$. This shows $(C_1, C_2) \in \mathrm{EC}(\cup, \times)$ if and only if $f(C_1, C_2) \in \mathrm{EC}(\cup, +)$. □

Proposition 10.34. $\mathrm{EC}(\cup, \times) \leq_{\mathrm{m}}^{\mathrm{p}} \mathrm{EC}(\cup, +)$

Proof. By Lemma 10.33, it suffices to construct a polynomial-time computable function that transforms a pair of $\{\cup, \times\}$-circuits (C_1, C_2) into a pair of $\{\cup, \times\}$-circuits (C_1', C_2') such that all inputs of C_1' and C_2' are positive and $(\mathrm{I}(C_1) = \mathrm{I}(C_2) \iff \mathrm{I}(C_1') = \mathrm{I}(C_2'))$.

Let C_1 and C_2 be two $\{\cup, \times\}$-circuits and assume without loss of generality that each of these circuits is connected. If neither C_1 nor C_2 has zero as input, then we are done by returning (C_1, C_2). If exactly one of the circuits has zero as input, then we know that this

circuit has zero in its output, but the other circuits has not. So we are done by returning two fixed non-equivalent circuits. From now on assume that both circuits have inputs that are zero and hence they have zero in their output.

Claim 10.35. There exists a polynomial-time computable function that on input of a $\{\cup, \times\}$-circuit C returns the following: If $\mathrm{I}(C) = \{0\}$ then the algorithm outputs "C computes the singleton $\{0\}$". Otherwise, the output is a $\{\cup, \times\}$-circuit C' such that $\mathrm{I}(C') = \mathrm{I}(C) - \{0\}$.

Proof of the claim. Observe that with an easy recursive algorithm, we can determine in polynomial time whether the set generated by a gate in C contains 0 and whether this set contains a positive number. If the output gate of C does not contain a positive number, then $\mathrm{I}(C) = \{0\}$ and we are done by returning "C computes the singleton $\{0\}$". Otherwise, the output gate of C contains at least one positive number. Let $\tilde{C}$ be the circuit that is obtained from C if each gate that does not generate a positive number is replaced by a new input gate with label 0. Clearly, $\mathrm{I}(\tilde{C}) = \mathrm{I}(C)$. So in $\tilde{C}$, each gate that is not an input gate and that is connected with the output gate generates a set that contains positive numbers. Therefore, in $\tilde{C}$, the direct successors of input gates that are zero and that are connected to the output gate must be $\cup$-gates. Let C' be the circuit that is obtained from $\tilde{C}$ by deleting all gates not connected to the output gate and all input gates that are zero (by doing the latter, the adjacent $\cup$-gates become edges). Observe that apart from 0, the gates in C' generate the same sets as the gates in $\tilde{C}$. This proves Claim 10.35. $\diamond$

Applying Claim 10.35 to C_1 and C_2. If both, C_1 and C_2, compute the singleton $\{0\}$, then we are done by returning two fixed equivalent circuits. If exactly one of the circuits computes the singleton $\{0\}$, then we are done by returning two fixed non-equivalent circuits. Otherwise, the algorithm in Claim 10.35 returns two circuits C_1' and C_2' such that $\mathrm{I}(C_1') = \mathrm{I}(C_1) - \{0\}$ and $\mathrm{I}(C_2') = \mathrm{I}(C_2) - \{0\}$. So we are done by returning (C_1', C_2'). $\square$

Theorem 10.36. The problems $\mathrm{EC}(\cup, +)$ and $\mathrm{EC}(\cup, \times)$ are $\leq_{\mathrm{m}}^{\log}$-complete for Π_2^{P}.

Proof. $\mathrm{EC}(\cup, \times)$ is Π_2^{P}-hard by Theorem 10.30 and $\mathrm{EC}(\cup, +)$ is Π_2^{P}-hard since the equivalence problem for expressions over $\{\cup, +\}$ is $\leq_{\mathrm{m}}^{\log}$-complete for Π_2^{P} by Theorem 10.10.2. Both problems are in Π_2^{P} because of Proposition 10.34 and Corollary 10.14. $\square$

10.3.4 More General Equivalence Problems

In this section we analyze equivalence problems which are more difficult to decide than the problems presented in the former section. For most of the problems here, we cannot give exact bounds but they are all PSPACE- or NEXP-hard. For the most general problem, $\mathrm{EC}(^{-}, \cup, \cap, +, \times)$, the best upper bound we can give is Δ_2, the sets (unboundedly) Turing-reducible to the halting problem. From the Propositions 10.15 and 10.17.1 it follows that this problem is equivalent to the respective membership problem, i. e. $\mathrm{EC}(^{-}, \cup, \cap, +, \times) \equiv_{\mathrm{m}}^{\log} \mathrm{MC}(^{-}, \cup, \cap, +, \times)$.

Every decision algorithm for $\mathrm{EC}(^{-}, \cup, \cap, +, \times)$ would enable us to automatically verify Goldbach's conjecture. This means that we run the algorithm on input of the circuit that formulates Goldbach's conjecture (this circuits is shown in the introduction) and the algorithm definitely tells us whether or not the conjecture is true. It is possible that $\mathrm{EC}(^{-}, \cup, \cap, +, \times)$ is undecidable, but at the moment, we cannot prove this.

Circuits over positive natural numbers will turn out useful to analyze the most general equivalence problem. These circuits can only have positive inputs and $^-$ is interpreted with respect to $\mathbb{N}^+$. Breunig [Bre07] showed that for such circuits the general membership problem, which we call $\mathrm{MC}_{\mathbb{N}^+}(^-, \cup, \cap, +, \times)$ is decidable in PSPACE. We will utilize this result to show that we can solve $\mathrm{EC}(^-, \cup, \cap, +, \times)$ if the evaluation algorithm has oracle access to the halting problem.

Theorem 10.37 ([Bre07])**.** $\mathrm{MC}_{\mathbb{N}^+}(^-, \cup, \cap, +, \times)$ is $\leq_{\mathrm{m}}^{\log}$-complete for PSPACE.

Lemma 10.38. There exists an oracle Turing machine M with oracle K (the halting problem) that on input of a $\{^-, \cup, \cap, +, \times\}$-circuit C over $\mathbb{N}$ outputs a $\{^-, \cup, \cap, +, \times\}$-circuit D over $\mathbb{N}^+$ and a set $Z \subseteq \{0\}$ such that $\mathrm{I}(C) = \mathrm{I}(D) \cup Z$.

Proof. Algorithm 10.2 defines M. Variables denoted by C represent circuits over $\mathbb{N}$, variables denoted by D represent circuits over $\mathbb{N}^+$, and variables denoted by Z represent subsets of $\{0\}$. On input of a circuit C over $\mathbb{N}$, the algorithm simulates C by a circuit D over $\mathbb{N}^+$ where a possible element 0 is stored in the separate set Z. More precisely, with help of recursive calls, the algorithm first determines $\mathbb{N}^+$-circuits D and sets Z that correspond to all direct predecessors of C's output gate, and then it joins the obtained circuits and sets in an appropriate way.

Algorithm 10.2: `isolate_zero(`C`)`

```
   Input : {⁻, ∪, ∩, +, ×}-circuit C = (V, E, g_C, α) over ℕ
   Output: {⁻, ∪, ∩, +, ×}-circuit D over ℕ⁺ and set Z ⊆ {0}
 1 if g_C is an input gate then
 2     if g_C has label l > 0 then return ({l}, ∅) else return (∅, {0});
   // here g_C is not an input gate
 3 if g_C has label ⁻ then
 4     let C' be the circuit obtained from C by defining g_C's direct predecessor to
       be the output gate of C' and deleting g_C;
 5     let (D, Z) := M(C');
 6     return (D̄, {0} − Z);
   // here g_C has a label from {∪, ∩, +, ×}
 7 let C_1 (C_2) be the circuit obtained from C by defining g_C's left (right) direct
   predecessor to be the new output gate and deleting g_C;
 8 let (D_1, Z_1) := M(C_1) and (D_2, Z_2) := M(C_2);
 9 if g_C has label ∩ then return (D_1 ∩ D_2, Z_1 ∩ Z_2);
10 if g_C has label ∪ then return (D_1 ∪ D_2, Z_1 ∪ Z_2);
11 if g_C has label + then
12     if Z_1 = {0} then D'_2 := D_2 else D'_2 := ∅;
13     if Z_2 = {0} then D'_1 := D_1 else D'_1 := ∅;
14     return ((D_1 + D_2) ∪ D'_1 ∪ D'_2, Z_1 ∩ Z_2);
15 if g_C has label × then
16     if Z_1 = {0} and I(D_2) ≠ ∅ (oracle access) then Z'_1 := {0} else Z'_1 := ∅;
17     if Z_2 = {0} and I(D_1) ≠ ∅ (oracle access) then Z'_2 := {0} else Z'_2 := ∅;
18     return (D_1 × D_2, Z'_1 ∪ Z'_2 ∪ (Z_1 ∩ Z_2));
```

First observe that whenever the algorithm makes a recursive call (lines 5 and 8), then it calls an instance smaller than C. So the algorithm terminates. Also, it is easy to observe that the algorithm outputs a $\{^-, \cup, \cap, +, \times\}$-circuit over $\mathbb{N}^+$ and a subset of $\{0\}$.

A straightforward induction over the circuit size shows that if the algorithm outputs (D, Z) then $\mathrm{I}(C) = \mathrm{I}(D) \cup Z$. For treating the $\times$-gates the algorithm needs oracle access at lines 16 and 17. By Theorem 10.37, the membership problem for $\{^-, \cup, \cap, +, \times\}$-circuits over $\mathbb{N}^+$ is decidable. Therefore, the non-emptiness problem is computably enumerable and hence can be answered with one query to our oracle, the halting problem. □

Theorem 10.39. $\mathrm{MC}(^-, \cup, \cap, +, \times), \mathrm{EC}(^-, \cup, \cap, +, \times) \in \Delta_2$, the class of sets (unboundedly) Turing-reducible to the halting problem.

Proof. We describe a Turing reduction from $\mathrm{MC}(^-, \cup, \cap, +, \times)$ to the halting problem. Let (C, n) be the input. With help of Lemma 10.38 we transform the $\{^-, \cup, \cap, +, \times\}$-circuit C over $\mathbb{N}$ into a $\{^-, \cup, \cap, +, \times\}$-circuit D over $\mathbb{N}^+$ and a set $Z \subseteq \{0\}$ such that $\mathrm{I}(C) = \mathrm{I}(D) \cup Z$. By doing so we make queries to the halting problem. If $n = 0$, then we accept if and only if $Z = \{0\}$. Otherwise, we accept if and only if $(D, n) \in \mathrm{MC}_{\mathbb{N}^+}(^-, \cup, \cap, +, \times)$. By Theorem 10.37, the latter is decidable in polynomial space. By the Propositions 10.15 and 10.17, $\mathrm{EC}(^-, \cup, \cap, +, \times) \equiv_{\mathrm{m}}^{\log} \mathrm{MC}(^-, \cup, \cap, +, \times)$. □

Proposition 10.40. The problem $\mathrm{EC}(^-, \cup, \cap, +, \times)$ is computably enumerable if and only if $\mathrm{EC}(^-, \cup, \cap, +, \times)$ is decidable. Furthermore, if $\mathrm{EC}(^-, \cup, \cap, +, \times) \in \mathrm{NEXP}$ then $\mathrm{NEXP} = \mathrm{coNEXP}$.

Proof. We have $\mathrm{EC}(^-, \cup, \cap, +, \times) \equiv_{\mathrm{m}}^{\log} \mathrm{MC}(^-, \cup, \cap, +, \times)$ by Propositions 10.15 and 10.17. Furthermore, $\mathrm{MC}(^-, \cup, \cap, +, \times) \leq_{\mathrm{m}}^{\log} \overline{\mathrm{MC}(^-, \cup, \cap, +, \times)}$, because of the $^-$-gates. The second assertion follows from the fact that $\mathrm{EC}(^-, \cup, \cap, +, \times)$ is NEXP-hard by Corollary 10.16. □

We note that the equivalence problems become decidable if we forbid the combination of $^-$-, $+$- and $\times$-gates.

Proposition 10.41.

1. $\mathrm{EC}(\cup, \cap, +, \times), \mathrm{EC}(\cup, +, \times) \in \mathrm{coNEXP}^{\mathrm{NP}}$.
2. $\mathrm{EC}(^-, \cup, \cap, \times), \mathrm{EC}(\cup, \cap, \times) \in \mathrm{PSPACE}$

Proof. For the first statement it suffices to consider $\mathrm{EC}(\cup, \cap, +, \times)$. We show that $\overline{\mathrm{EC}(\cup, \cap, +, \times)} \in \mathrm{NEXP}^{\mathrm{MF}(\cup, \cap, +, \times)}$, where $\mathrm{MF}(\cup, \cap, +, \times) \in \mathrm{NP}$ [MW07] is the restriction of the membership problem for circuits to circuits where each gate has outdegree at most one and thus is an expression.

For given circuits C_1 and C_2, the nondeterministic exponential time oracle Turing machine chooses nondeterministically a number n between 0 and $\max\{2^{2^{2|C_1|}}, 2^{2^{2|C_2|}}\}$. Then it unfolds the circuits C_1 and C_2 to expressions F_1 and F_2 (cf. Lemma 10.9.4). Both F_1 and F_2 are at most exponentially large. The machine accepts if and only if the oracle gives different answers to the queries (F_1, n) and (F_2, n).

By Proposition 10.12, the maximal value in the output of a $\{\cup, \cap, +, \times\}$-circuit C is bounded by $2^{2^{2|C|}}$. Hence, the circuits are not equivalent if and only if there is some number n between 0 and $\max\{2^{2^{2|C_1|}}, 2^{2^{2|C_2|}}\}$ such that $n \in \mathrm{I}(C_1) \iff n \notin \mathrm{I}(C_2)$. Thus the machine accepts the language $\overline{\mathrm{EC}(\cup, \cap, +, \times)}$ and therefore $\mathrm{EC}(\cup, \cap, +, \times) \in \mathrm{coNEXP}^{\mathrm{NP}}$.

For the second statement, note that $\mathrm{EC}(^-, \cup, \cap, \times) \in \mathrm{PSPACE}$ follows from Proposition 10.17.1 and [MW07]. □

Testing equivalence for $\{\cup, +, \times\}$-circuits is likely to be more difficult than testing membership: While the membership problem is PSPACE-complete [MW07], we can show that the equivalence problem is NEXP-hard. For the membership problem, it suffices to compute numbers in the length of the target number, as numbers can only become smaller when multiplied by 0. Intuitively, the difficulty when testing equivalence for $\{\cup, +, \times\}$-circuits is that we have to deal with very large (up to exponential in length) numbers.

Travers observed that a similar effect occurs when testing membership for $\{\cup, +, \times\}$-circuits over the *integers*: By describing a generic reduction, it was shown that the problem $\mathrm{MC}_{\mathbb{Z}}(\cup, +, \times)$ is NEXP-complete [Tra06]. With minor modifications this proof also shows that $\mathrm{EC}(\cup, +, \times)$ is $\leq_{\mathrm{m}}^{\log}$-hard for NEXP.

Corollary 10.42. The problem $\mathrm{EC}(\cup, +, \times)$ is $\leq_{\mathrm{m}}^{\log}$-hard for NEXP.

10.4 Satisfiability Problems

In contrast to the equivalence problems, satisfiability problems are always a direct generalization of the respective membership problems and thus the lower bounds are immediately applicable here. Concerning upper bounds, it is obvious that $\mathrm{SC}(\mathcal{O})$ is always an unbounded projection of $\mathrm{MC}(\mathcal{O})$. For some sets of operations (namely $\mathcal{O} \subseteq \{^-, \cup, \cap, +\}$ and $\mathcal{O} \subseteq \{\cup, +, \times\}$), we can show that the projection is actually polynomially bounded.

Proposition 10.43. The following results are immediate consequences of the known results about membership problems.

1. $\mathrm{SC}(^-, \cup, \cap, +, \times)$, $\mathrm{SC}(^-, \cup, \cap, \times)$, $\mathrm{SC}(\cup, \cap, +, \times)$, $\mathrm{SC}(\cap, +, \times) \in \Sigma_2$.
2. $\mathrm{SC}(^-, \cup, \cap, +)$, $\mathrm{SC}(\cup, \cap, +)$, $\mathrm{SC}(\cup, \cap, \times)$, $\mathrm{SC}(^-, \cup, \cap, \times)$ and $\mathrm{SC}(\cup, +, \times)$ are $\leq_{\mathrm{m}}^{\log}$-hard for PSPACE.
3. $\mathrm{SC}(\cup, \times)$ is $\leq_{\mathrm{m}}^{\log}$-hard for NP.
4. $\mathrm{SC}(\cap)$ and $\mathrm{SC}(\cup)$ are $\leq_{\mathrm{m}}^{\log}$-complete for NL.
5. $\mathrm{SC}(\times)$ is $\leq_{\mathrm{m}}^{\log}$-hard for NL.
6. $\mathrm{SC}(\cup, \cap)$ is $\leq_{\mathrm{m}}^{\log}$-complete for P.

Proof. All hardness results directly follow from results by McKenzie and Wagner [MW07]. For the first statement, we know that $\mathrm{MC}(^-, \cup, \cap, +, \times) \in \Delta_2$ by Theorem 10.39. Since $\mathrm{SC}(^-, \cup, \cap, +, \times)$ is an unrestricted projection of $\mathrm{MC}(^-, \cup, \cap, +, \times)$, it lies in $\exists \cdot \Delta_2 = \Sigma_2$. For the upper bounds where $\mathcal{O} \subseteq \{\cup, \cap\}$, note that on input (C, b), it suffices to run the algorithm for the respective $\mathrm{MC}(\mathcal{O})$-problem by setting all variable inputs to b. □

Lemma 10.44. Let C be a circuit over the operations $\mathcal{O} \subseteq \{^-, \cup, \cap, +, \times\}$ with exactly n unassigned inputs. For $b \in \mathbb{N}$, $x_1, \ldots, x_n \in \mathbb{N}$ and $c \leq b$ it holds that

1. if $\mathcal{O} \subseteq \{^-, \cup, \cap, +\}$, then
   $c \in \mathrm{I}(C(x_1, \ldots, x_n)) \iff c \in \mathrm{I}(C(\min(x_1, b+1), \ldots, \min(x_n, b+1)))$.

2. if $\mathcal{O} \subseteq \{\cup, +, \times\}$, then
   $c \in \mathrm{I}(C(x_1, \ldots, x_n)) \Longrightarrow c \in \mathrm{I}(C(\min(x_1, b+1), \ldots, \min(x_n, b+1)))$.

Proof. We show both parts by induction over the number of direct and indirect predecessors of the output gate and, for the sake of brevity, use the notations $\tilde{x} := (x_1, x_2, \ldots, x_n)$ and $\min(\tilde{x}, b+1) := (\min(x_1, b+1), \min(x_2, b+1), \ldots, \min(x_n, b+1))$.

1. For the induction base consider C to be a circuit where the output gate is the first input gate and let $c \leq b$. If this input gate is assigned, the assertion is obviously true. Otherwise, the following equivalence holds:

$$\begin{aligned} c \in \mathrm{I}(C(\tilde{x})) &\iff c = x_1 \iff c = \min(x_1, b+1) \\ &\iff c \in \mathrm{I}(C(\min(\tilde{x}, b+1))) \end{aligned}$$

For the induction step, let C be a circuit whose output gate g has at least one direct predecessor, let $b \in \mathbb{N}$ and $c \leq b$. The cases where g has the label $^-$, $\cup$ or $\cap$ are straightforward. So let g have the label $+$ and the direct predecessors u and v. Let C_1 (resp., C_2) be the circuit that has u (resp., v) as output gate and is otherwise equal to C.

Note that $c \in \mathrm{I}(C(\tilde{x}))$ if and only if there exist $s \in \mathrm{I}(C_1(\tilde{x}))$ and $t \in \mathrm{I}(C_2(\tilde{x}))$ such that $s, t \leq c \leq b$ and $s + t = c$. Thus, by the induction hypothesis, for all these s, t it holds that $s \in \mathrm{I}(C_1(\tilde{x}))$ and $t \in \mathrm{I}(C_2(\tilde{x}))$ if and only if $s \in \mathrm{I}(C_1(\min(\tilde{x}, b+1)))$ and $t \in \mathrm{I}(C_2(\min(\tilde{x}, b+1)))$. We obtain $c \in \mathrm{I}(C(\tilde{x}))$ if and only if $c \in \mathrm{I}(C_1(\min(\tilde{x}, b+1)) + C_2(\min(\tilde{x}, b+1))) = \mathrm{I}(C(\min(\tilde{x}, b+1)))$.

2. We argue as above, noting that here we merely have to show one implication. The only new consideration occurs in the induction step where g has label $\times$ and $c = 0$ ($c > 0$ is treated similarly to the $+$ case). Then there must be, without loss of generality, some $t \in \mathrm{I}(C_2(\tilde{x}))$ and $0 \in \mathrm{I}(C_1(\tilde{x}))$. By the induction hypothesis, we have $0 \in \mathrm{I}(C_1(\min(\tilde{x}, b+1)))$. Since no gate in a circuit over $\{\cup, +, \times\}$ computes the empty set, we clearly have $\mathrm{I}(C_2(\min(\tilde{x}, b+1))) \neq \emptyset$ and thus $c = 0 \in \mathrm{I}(C(\min(\tilde{x}, b+1)))$. □

Proposition 10.45. For any set of operations $\mathcal{O} \subseteq \{^-, \cup, \cap, +\}$ or $\mathcal{O} \subseteq \{\cup, +, \times\}$ and any complexity class $\mathcal{C}$ closed under $\leq_{\mathrm{m}}^{\log}$ it holds that

$$\mathrm{MC}(\mathcal{O}) \in \mathcal{C} \Longrightarrow \mathrm{SC}(\mathcal{O}) \in \exists^{\mathrm{p}} \cdot \mathcal{C}.$$

Proof. Let C be a circuit over $\mathcal{O}$ with exactly n unassigned inputs and let $b \in \mathbb{N}$. Lemma 10.44 shows that if there is an input assignment that generates b, there is also an input assignment bounded by $b+1$ that generates b. Hence it suffices to check if $b \in \mathrm{I}(C(x_1, \ldots, x_n))$ for $x_1, \ldots, x_n \in \{0, 1, \ldots, b+1\}$. Since the length of $(x_1, \ldots, x_n)$ is polynomial in the input length as long as $x_i \leq b+1$ for all $1 \leq i \leq n$, the assertion follows. □

Corollary 10.46. It holds that

1. $\mathrm{SC}(^-, \cup, \cap, +)$, $\mathrm{SC}(\cup, \cap, +)$ and $\mathrm{SC}(\cup, +, \times)$ are in PSPACE.
2. $\mathrm{SC}(^-, \cup, \cap)$, $\mathrm{SC}(\cap, +)$, $\mathrm{SC}(\cup, \times)$, $\mathrm{SC}(\cup, +)$, $\mathrm{SC}(+)$ and $\mathrm{SC}(+, \times)$ are in NP.

Proof. Since $\exists^{\mathrm{p}} \cdot \mathrm{PSPACE} = \mathrm{PSPACE}$ and $\exists^{\mathrm{p}} \cdot \mathrm{NP} = \mathrm{NP}$, the statement follows from Proposition 10.45 and the results by McKenzie and Wagner [MW07]. □

10.4.1 Undecidable Problems

We have seen in Lemma 10.9.3 that circuits with gates + and × can be used to represent multivariate polynomials. The presence of ∩ then allows us to translate the solvability of Diophantine equations into the satisfiability of circuits. Hence these satisfiability problems are undecidable. In particular, $\mathrm{SC}(\cap, +, \times)$ is not a polynomially bounded projection of its corresponding membership problem.

Theorem 10.47. $\mathrm{SC}(\cap, +, \times)$ is $\leq_{\mathrm{m}}^{\log}$-hard for Σ_1 and thus undecidable.
The same is true for $\mathrm{SC}(\cup, \cap, +, \times)$ and $\mathrm{SC}(^{-}, \cup, \cap, +, \times)$.

Proof. Let $A \in \Sigma_1$, i. e. A is computably enumerable. By the Davis-Putnam-Robinson-Matiyasevich theorem [DPR61, Mat70], there is a multivariate polynomial $p(x_0, x_1, \ldots, x_n)$ with integer coefficients such that for all $x_0 \in \mathbb{N}$

$$x_0 \in A \iff \exists x_1, \ldots, x_n \in \mathbb{N} \colon p(x_0, x_1, \ldots, x_n) = 0.$$

By moving monomials with negative coefficients to the right-hand side, we equivalently transform the equation $p(x_0, x_1, \ldots, x_n) = 0$ to $l(x_0, x_1, \ldots, x_n) = r(x_0, x_1, \ldots, x_n)$ such that all coefficients in l and r are positive. According to Lemma 10.9.3, there are $\{+, \times\}$-circuits C_l and C_r with $n+1$ unassigned inputs such that $\mathrm{I}(C_l(x_0, x_1, \ldots, x_n)) = \{l(x_0, x_1, \ldots, x_n)\}$ and $\mathrm{I}(C_r(x_0, x_1, \ldots, x_n)) = \{r(x_0, x_1, \ldots, x_n)\}$.

We now define a function f that maps natural numbers to circuits in the following way: On input x_0 it constructs a circuit C' with n unassigned inputs such that $C'(x_1, \ldots, x_n) := 0 \times (C_l(x_0, x_1, \ldots, x_n) \cap C_r(x_0, x_1, \ldots, x_n))$. Note that f is easily computable in logarithmic space since it only has to assign the label x_0 to a fixed circuit.

We then get

$$\begin{aligned} x_0 \in A &\iff \exists x_1, \ldots, x_n \in \mathbb{N} \colon p(x_0, x_1, \ldots, x_n) = 0 \\ &\iff \exists x_1, \ldots, x_n \in \mathbb{N} \colon 0 \in \mathrm{I}(C'(x_1, \ldots, x_n)) \\ &\iff (C', 0) \in \mathrm{SC}(\cap, +, \times). \end{aligned}$$

This completes the reduction. □

10.4.2 Circuits with both Arithmetic and Set Operations

In this section, we will show upper and lower bounds for decidable satisfiability problems for circuits that may use arithmetic and set operations at the same time. Among these, the problem $\mathrm{SC}(\cap, \times)$ has an interesting property. In contrast to most other NP-complete problems, here proving the membership in NP is more difficult than proving the hardness for NP. We will use the (nontrivial) result that integer programming belongs to NP to show that certain systems of monomial equations can be solved in NP. This result is then used to finally establish $\mathrm{SC}(\cap, \times) \in \mathrm{NP}$. We start by defining the mentioned problem concerning systems of monomial equations.

Definition 10.48. Informally, the problem MonEq asks if systems of equations of the form

$$\begin{aligned} x^6 z^7 &= 5^9 y^3 z^2 \\ y z^2 &= 2^3 x^4 \\ x^2 y^4 z^3 &= 3^{11} \end{aligned}$$

are solvable over the natural numbers. Formally, we have the following definition (where we set $0^0 := 1$):

$$\begin{aligned}\textsc{MonEq} := \Big\{(A,B,C,D) \mid{}& n,m \geq 1, A = (a_{i,j}) \in \mathbb{N}^{m\times n}, B = (b_{i,j}) \in \mathbb{N}^{m\times n},\\ & C = (c_1,\dots,c_m) \in \mathbb{N}^m, D = (d_1,\dots,d_m) \in \mathbb{N}^m,\\ & \text{and there exist } x_1,\dots,x_n \in \mathbb{N} \text{ such that}\\ & \text{for all } 1 \leq i \leq m\colon \prod_{j=1}^{n} x_j^{a_{i,j}} = c_i^{d_i} \cdot \prod_{j=1}^{n} x_j^{b_{i,j}}\Big\}\end{aligned}$$

Note that this definition neither allows constant factors on the left-hand side of equations nor allows products of constant factors like $2^{91} \cdot 3^{93} \cdot 5^{97}$. However, such factors can be easily expressed by using additional variables. For example, the equation $7^3 \cdot 15^{70} \cdot x^5y^7 = 3^7z^3$ can be equivalently transformed into the following system.

$$\begin{aligned} a &= 7^3\\ b &= 15^{70}\\ abx^5y^7 &= 3^7z^3\end{aligned}$$

We show that systems of monomial equations can be solved in nondeterministic polynomial time. Our proof transforms the original problem MonEq to a more restricted version. Then we show the latter to be in NP where we use the fact that integer programming belongs to NP.

Lemma 10.49. MonEq $\in$ NP.

Proof. We start with the definition of a variant of MonEq that restricts to positive constant factors and positive solutions.

$$\begin{aligned}\textsc{MonEq}^+ := \Big\{(A,B,C,D) \mid{}& n,m \geq 1, A = (a_{i,j}) \in \mathbb{N}^{m\times n}, B = (b_{i,j}) \in \mathbb{N}^{m\times n},\\ & C = (c_1,\dots,c_m) \in (\mathbb{N}^+)^m, D = (d_1,\dots,d_m) \in \mathbb{N}^m,\\ & \text{and there exist } x_1,\dots,x_n \in \mathbb{N}^+ \text{ such that}\\ & \text{for all } 1 \leq i \leq m\colon \prod_{j=1}^{n} x_j^{a_{i,j}} = c_i^{d_i} \cdot \prod_{j=1}^{n} x_j^{b_{i,j}}\Big\}\end{aligned}$$

Assume for the moment that we have shown MonEq$^+ \in$ NP. Under this assumption we can describe a nondeterministic polynomial-time algorithm that accepts MonEq. The input is a MonEq instance (A,B,C,D). First, we nondeterministically guess the variables that will be equal to 0, i. e., we guess a set $I \subseteq \{1,\dots,n\}$ and demand that $x_i = 0$ for all $i \in I$ and that $x_j > 0$ for all $j \notin I$. This allows us to determine whether or not a certain side of an equation is 0. If we detect an inconsistency (one side is equal to zero, the other greater than zero), we reject. Otherwise, we remove all equations where both sides equal zero and arrive at a MonEq$^+$ instance which by assumption can be solved in NP. This shows MonEq $\in$ NP. So it remains to prove MonEq$^+ \in$ NP.

We define a variant of MonEq$^+$ that restricts to the case where all constant factors and all components of the solution are powers of the same prime p.

$$\begin{aligned}\textsc{MonEq}^{\text{p}} := \Big\{(A,B,p,D) \mid\, & n,m \geq 1, A = (a_{i,j}) \in \mathbb{N}^{m\times n}, B = (b_{i,j}) \in \mathbb{N}^{m\times n},\\ & D = (d_1,\ldots,d_m) \in \mathbb{N}^m, p \text{ is a prime},\\ & \text{and there exist } x_1,\ldots,x_n \in \{p^r \mid r \in \mathbb{N}\}\\ & \text{such that for all } 1 \leq i \leq m\\ & \prod_{j=1}^{n} x_j^{a_{i,j}} = p^{d_i} \cdot \prod_{j=1}^{n} x_j^{b_{i,j}}\Big\}\end{aligned}$$

Assume for the moment that we have shown MonEq$^{\text{p}} \in$ NP. Under this assumption we show that MonEq$^+ \in$ NP. Let the MonEq$^+$ instance (A,B,C,D) be our input and let $p_1,\ldots,p_l$ be the primes that appear in the prime factorization of the numbers $c_1,\ldots,c_m$ and let $e_{k,i}$ be the exponents such that $c_i = \prod_{k=1}^{l} p_k^{e_{k,i}}$ for $1 \leq i \leq m$. Let us consider the following equivalence.

$$\begin{aligned}(A,B,C,D) \in \textsc{MonEq}^+ \iff\ & \forall 1 \leq k \leq l\, \exists x_{k,1},\ldots,x_{k,n} \in \{p_k^r \mid r \in \mathbb{N}\}\\ & \forall 1 \leq i \leq m\colon \prod_{j=1}^{n} x_{k,j}^{a_{i,j}} = p_k^{e_{k,i}\cdot d_i} \cdot \prod_{j=1}^{n} x_{k,j}^{b_{i,j}}\end{aligned} \tag{10.2}$$

The implication from right to left is easy to see, since with $x_j := \prod_{k=1}^{l} x_{k,j}$ we obtain a solution $x_1,\ldots,x_n$ for the MonEq$^+$ instance (A,B,C,D). So let us consider the implication from left to right and let $x_1,\ldots,x_n$ be a solution for (A,B,C,D). This means that all prime factors of $x_1,\ldots,x_n$ are from $\{p_1,\ldots,p_l\}$. Let $x_{k,j}$ be the number that is obtained from x_j by removing all prime factors different from p_k. Because of the unique prime decomposition of positive integers, $x_{k,1},\ldots,x_{k,n}$ is a solution for the system on the right-hand side of (10.2). This proves the equivalence (10.2).

This shows that MonEq$^+$ is conjunctively truth-table reducible to MonEq$^{\text{p}}$ in polynomial time. By our assumption, the latter is in NP and therefore, we obtain MonEq$^+ \in$ NP. So it remains to prove MonEq$^{\text{p}} \in$ NP.

The definition of MonEq$^{\text{p}}$ demands that each element x_j of the solution can be written as $x_j = p^{e_j}$ for a suitable $e_j \in \mathbb{N}$. We obtain

$$\begin{aligned}(A,B,p,D) \in \textsc{MonEq}^{\text{p}} \iff\ & p \text{ is prime and there exist } e_1,\ldots,e_n \in \mathbb{N}\\ & \text{such that for all } 1 \leq i \leq m\\ & \prod_{j=1}^{n} p^{e_j a_{i,j}} = p^{d_i} \cdot \prod_{j=1}^{n} p^{e_j b_{i,j}}\\ \iff\ & p \text{ is prime and there exist } e_1,\ldots,e_n \in \mathbb{N}\\ & \text{such that for all } 1 \leq i \leq m\\ & \sum_{j=1}^{n} e_j a_{i,j} = d_i + \sum_{j=1}^{n} e_j b_{i,j}.\end{aligned}$$

The last statement can be expressed by the following integer program in the variables $e_1,\ldots,e_n$.

$$\begin{aligned}
\sum_{j=1}^{n} e_j a_{i,j} &\leq d_i + \sum_{j=1}^{n} e_j b_{i,j} && \text{for } 1 \leq i \leq m \\
\sum_{j=1}^{n} e_j a_{i,j} &\geq d_i + \sum_{j=1}^{n} e_j b_{i,j} && \text{for } 1 \leq i \leq m \\
e_j &\geq 0 && \text{for } 1 \leq j \leq n
\end{aligned}$$

For such systems of inequalities, the existence of integer solutions can be verified in NP [Kar72]. This shows $\textsc{MonEq}^{\text{p}} \in \text{NP}$ and finishes the proof of the lemma. □

Utilizing the fact that systems of monomial equations can be solved in nondeterministic polynomial time we now show that $\text{SC}(\cap, \times)$ belongs to NP. Observe that this is nontrivial, since the smallest satisfying assignment of an $\{\cap, \times\}$-circuit can be exponentially long because of repeated squaring.

Theorem 10.50. $\text{SC}(\cap, \times) \in \text{NP}$

Proof. We describe a nondeterministic polynomial-time algorithm for $\text{SC}(\cap, \times)$ on input (C, d). Without loss of generality we may assume that each gate in C is connected to the output gate, the nodes $1, \ldots, m$ are the unassigned input gates and the nodes $m+1, \ldots, m+n$ are the assigned input gates with labels $b_1, \ldots, b_n$. To each gate g of C with direct predecessors g_1 and g_2 (if any) we recursively attach a monomial $\beta(g)$ over $x_1, \ldots, x_{m+n}$ in the following way:

$$\beta(g) = \begin{cases} x_i & \text{if } g = i \leq m+n, \\ \beta(g_1) \cdot \beta(g_2) & \text{if } g \text{ is a } \times\text{-gate}, \\ \beta(g_1) & \text{if } g \text{ is an } \cap\text{-gate}. \end{cases}$$

During the attachment process, we simplify all products by combining multiple occurrences of the same variable. Using β, we generate a system of monomial equations:

$$\begin{aligned}
\beta(g_1) &= \beta(g_2) && \text{for each } \cap\text{-gate with direct predecessors } g_1 \text{ and } g_2, \\
x_{m+i} &= b_i && \text{for } 1 \leq i \leq n, \\
\beta(g_C) &= d.
\end{aligned}$$

Where g_C is the output gate of C and $b_i \in \mathbb{N}$ is the label of the unassigned input gate $m+i$. Our algorithm accepts if and only if the obtained system of monomial equations has a solution within the natural numbers. By Lemma 10.49, the described algorithm is a nondeterministic polynomial-time algorithm. So it remains to argue for the correctness of this algorithm.

For a gate g, let $\beta(g)(a_1, \ldots, a_m, b_1, \ldots, b_n)$ denote the number that is obtained when the monomial $\beta(g)$ is evaluated at $x_1 = a_1$, …, $x_m = a_m$, $x_{m+1} = b_1$, …, $x_{m+n} = b_n$. A straightforward induction on the structure of C yields the following.

Claim 10.51. For all $a_1, \ldots, a_m \in \mathbb{N}$ and all gates g of the circuit $C(a_1, \ldots, a_m)$ the gate g either computes $\emptyset$ or the set $\{\beta(g)(a_1, \ldots, a_m, b_1, \ldots, b_n)\}$.

We show that the algorithm accepts (C, d) if and only if $(C, d) \in \mathrm{SC}(\cap, \times)$. Assume our algorithm accepts on input (C, d). So there exist $a_1, \ldots, a_m$ such that $a_1, \ldots, a_m, b_1, \ldots, b_n$ is a solution for the constructed system of monomial equations. Let $C' := C(a_1, \ldots, a_m)$. We first show that $\mathrm{I}(C') \neq \emptyset$. Suppose to the contrary that $\mathrm{I}(C') = \emptyset$. Then there exists an $\cap$-gate g with direct predecessors g_1 and g_2 such that g_1 and g_2 compute different but non-empty sets in C'. By Claim 10.51, g_1 computes $\{\beta(g_1)(a_1, \ldots, a_m, b_1, \ldots, b_n)\}$ and g_2 computes $\{\beta(g_2)(a_1, \ldots, a_m, b_1, \ldots, b_n)\}$ in C'. The equation $\beta(g_1) = \beta(g_2)$ appears in our system of monomial equations. Hence it holds that $\beta(g_1)(a_1, \ldots, a_m, b_1, \ldots, b_n) = \beta(g_2)(a_1, \ldots, a_m, b_1, \ldots, b_n)$ and hence g_1 and g_2 compute the same set in C'. This contradicts our assumption and thus $\mathrm{I}(C') \neq \emptyset$.

This means that by Claim 10.51, $\mathrm{I}(C') = \{\beta(g_C)(a_1, \ldots, a_m, b_1, \ldots, b_n)\}$ where g_C is the output gate. The equation $\beta(g_C) = d$ appears in the system of monomial equations, so $\mathrm{I}(C') = \{d\}$ and hence $(C, d) \in \mathrm{SC}(\cap, \times)$.

Conversely, assume now that $(C, d) \in \mathrm{SC}(\cap, \times)$, i. e., there exist $a_1, \ldots, a_m \in \mathbb{N}$ such that $\mathrm{I}(C(a_1, \ldots, a_m)) = \{d\}$. We show that $x_1 = a_1$, …, $x_m = a_m$, $x_{m+1} = b_1$, …, $x_{m+n} = b_n$ is a solution for the system of monomial equations that is constructed by the algorithm, which implies that the algorithm accepts on input (C, d). In the circuit $C' := C(a_1, \ldots, a_m)$, each $\cap$-gate g computes a non-empty set. So if g_1 and g_2 are the direct predecessors of g, then g, g_1 and g_2 all compute the same set in C'. From Claim 10.51 it follows that $\beta(g_1)(a_1, \ldots, a_m, b_1, \ldots, b_n) = \beta(g_2)(a_1, \ldots, a_m, b_1, \ldots, b_n)$. So all equations of the form $\beta(g_1) = \beta(g_2)$ are satisfied. Moreover, the additional equations of the form $x_{m+i} = b_i$ are trivially satisfied by our solution. From $\mathrm{I}(C') = \{d\}$ and from the claim it follows that $\beta(g_C)(a_1, \ldots, a_m, b_1, \ldots, b_n) = d$ where g_C is the output gate of C. This shows that all equations of our system are satisfied by the solution $(a_1, \ldots, a_m, b_1, \ldots, b_n)$ and the algorithm accepts. □

Finally, we establish the lower bound for $\mathrm{SC}(\cap, \times)$.

Theorem 10.52. $\mathrm{SC}(\cap, \times)$ is $\leq_{\mathrm{m}}^{\log}$-hard for NP.

Proof. We describe a $\leq_{\mathrm{m}}^{\log}$-reduction from 3SAT to $\mathrm{SC}(\cap, \times)$. Let the input for the reduction be a Boolean formula in 3-CNF over the variables $x_1, \ldots, x_n$ consisting of m clauses. Each clause is a disjunction of three literals $l_{i,1}, l_{i,2}, l_{i,3} \in \{x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}\}$ for $1 \leq i \leq m$.

The reduction function constructs a circuit C with $2n + 4m$ unassigned inputs $x_i, \overline{x_i}$, $y_j, \overline{y_j}$, $z_j, \overline{z_j}$ for $1 \leq i \leq n$ and $1 \leq j \leq m$. This circuit computes the intersection of several subcircuits. Each of these subcircuits computes one of the following expressions over the unassigned inputs.

1. $(x_i \times \overline{x_i}) \times (x_i \times \overline{x_i}) \times (x_i \times \overline{x_i})$	for $1 \leq i \leq n$
2. $(y_j \times \overline{y_j}) \times (y_j \times \overline{y_j}) \times (y_j \times \overline{y_j})$	for $1 \leq j \leq m$
3. $(z_j \times \overline{z_j}) \times (z_j \times \overline{z_j}) \times (z_j \times \overline{z_j})$	for $1 \leq j \leq m$
4. $l_{j,1} \times l_{j,2} \times l_{j,3} \times y_j \times z_j$	for $1 \leq j \leq m$

Note that $l_{j,k} \in \{x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}\}$ are the literals of the clauses. Finally, the reduction function outputs $(C, 8)$.

Observe that the reduction can be computed in logarithmic space. We now argue that it shows 3SAT $\leq_{\mathrm{m}}^{\log}$ $\mathrm{SC}(\cap, \times)$. First, assume that the Boolean formula belongs to 3SAT.

So there exist assignments $a_1, \ldots, a_n \in \{0, 1\}$ for the n variables that satisfy all clauses. We assign the numbers 2^{a_i} to the input gates x_i and 2^{1-a_i} to the input gates $\overline{x_i}$. By doing this, we make sure that all subcircuits of type 1 compute $\{8\}$. Moreover, we assign the following numbers to the input gates y_j, $\overline{y_j}$, z_j, and $\overline{z_j}$:

Assume that the jth clause contains exactly k ($1 \leq k \leq 3$) literals that evaluate to true with respect to the assignment $a_1, \ldots, a_n$. Let $b, c \in \{0, 1\}$ such that $b + c + k = 3$. Assign the value 2^a to y_i, 2^{1-a} to $\overline{y_i}$, 2^b to z_i, 2^{1-b} to $\overline{z_i}$. Observe that this assignment makes sure that all of the subcircuits evaluate to $\{8\}$. Hence, the output gate of C evaluates to $\{8\}$ which shows $(C, 8) \in \mathrm{SC}(\cap, \times)$.

Now assume that $(C, 8) \in \mathrm{SC}(\cap, \times)$; we will show that the corresponding Boolean formula is satisfiable. By assumption, all of the subcircuits evaluate to $\{8\}$. For the subcircuits of type 1, this implies that for each $1 \leq i \leq n$, one of x_i and $\overline{x_i}$ is assigned 2 while the other one is assigned 1. Similarly for the subcircuits of type 2 and 3, it holds that for each $1 \leq j \leq m$, one of y_j and $\overline{y_j}$ (resp., z_j and $\overline{z_j}$) is assigned 2 while the other one is assigned 1. For $1 \leq i \leq n$, if x_i is assigned 2, then let $a_i = 1$, otherwise let $a_i = 0$. The fact that each subcircuit of type 3 evaluates to $\{8\}$ shows that for each $1 \leq j \leq m$, $l_{j,1} \times l_{j,2} \times l_{j,3} \times y_j \times z_j$ evaluates to $\{8\}$. Since y_j and z_j are both at most 2, at least one of $l_{j,1}$, $l_{j,2}$ or $l_{j,3}$ must be assigned 2. Hence, the ith clause contains at least one literal that evaluates to true with respect to the assignment $a_1, \ldots, a_n$. This shows that the Boolean formula is satisfiable. Therefore, $3\textsc{Sat} \leq_{\mathrm{m}}^{\log} \mathrm{SC}(\cap, \times)$ and hence $\mathrm{SC}(\cap, \times)$ is $\leq_{\mathrm{m}}^{\log}$-hard for NP. □

The next corollary shows that we can utilize the algorithm presented in Theorem 10.50 which solves the satisfiability problem for $\{\cap, \times\}$-circuits in NP also to solve the problem for $\{\cup, \cap, \times\}$-circuits: However, to cope with the $\cup$-gates we first have to unfold the circuit into an expression. This can cause an exponential blow up in the size of the circuit.

Corollary 10.53. $\mathrm{SC}(\cup, \cap, \times) \in \mathrm{NEXP}$.

Proof. We describe a nondeterministic exponential time algorithm that accepts the problem $\mathrm{SC}(\cup, \cap, \times)$. Let C be a $\{\cup, \cap, \times\}$-circuit and $b \in \mathbb{N}$. We can assume that each gate in C is connected to the output gate. In exponential time, we can unfold the circuit into a (possibly exponentially larger) circuit C' where all gates except for the variable input gates have outdegree at most 1 and it holds for all $x_1, \ldots, x_n \in \mathbb{N}$ that

$$\mathrm{I}(C(x_1, \ldots, x_n)) = \mathrm{I}(C'(x_1, \ldots, x_n))$$

by Lemma 10.9.4. Let $g_\cup$ be the set of $\cup$-gates in C'. We now nondeterministically choose a total function $f \colon g_\cup \to \{1, 2\}$ ($\# g_\cup$ is at most exponential in the input length). Using this function we construct a circuit C_f in the following way: For each $\cup$-gate g with direct predecessors g_1 and g_2 we delete g and its incident edges. Furthermore, if g has a direct successor s, we add an edge from $g_{f(g)}$ to s. Finally, if g is the output gate of C', then $g_{f(g)}$ is the new output gate of C_f. Observe that C_f is an $\{\cap, \times\}$-circuit. By a straightforward induction it can be shown that

$$(C', b) \in SC(\cup, \cap, \times) \iff \exists f \colon g_\cap \to \{1, 2\} \colon (C_f, b) \in \mathrm{SC}(\cap, \times)$$

Hence it remains to call a suitable algorithm for $\mathrm{SC}(\cap, \times)$ on (C_f, b). Since by Theorem 10.50, $\mathrm{SC}(\cap, \times) \in \mathrm{NP}$, the whole procedure described above can be performed in nondeterministic exponential time and thus $\mathrm{SC}(\cup, \cap, \times) \in \mathrm{NEXP}$. □

10.4.3 Circuits with either Arithmetic or Set Operations

We now discuss that, perhaps somewhat surprisingly, SC(×) is likely to be easier than SC(+). More precisely, we show that SC(×) ∈ UP ∩ coUP and prove SC(+) to be NP-complete. For the lower bound of SC(+) we need the following variant of the KNAPSACK-problem which is known to be NP-complete [GJ79, MP10].

$$\text{KNAPSACK}' := \{(v_1, \ldots, v_n, b) \mid n \geq 1, v_1, \ldots v_n, b \geq 1 \text{ and there exist } u_1, \ldots, u_n \in \mathbb{N} \text{ such that } \sum_{i=1}^{n} u_i v_i = b\}$$

Theorem 10.54. SC(+) and SC(+, ×) are $\leq_{\mathrm{m}}^{\log}$-complete for NP.

Proof. We describe a reduction from KNAPSACK′ to SC(+). On input $(v_1, \ldots, v_n, b)$, we use Lemma 10.9.2 to construct {+}-circuits that compute $\{v_i \cdot x_i\}$ with variable input gate x_i for $1 \leq i \leq n$. We combine them to a single {+}-circuit C such that $\mathrm{I}(C(x_1, \ldots, x_n)) = \{v_1 x_1 + \cdots + v_n x_n\}$ for all $x_1, \ldots, x_n \in \mathbb{N}$. The reduction outputs (C, b). Observe that $(v_1, \ldots, v_n, b) \in$ KNAPSACK′ if and only if $(C, b) \in$ SC(+). So SC(+) and SC(+, ×) are NP-hard. The membership in NP follows from Corollary 10.46. □

By MC(×) ∈ NL [MW07] and Proposition 10.45, it is immediately clear that SC(×) ∈ NP. We now prove the better upper bound UP ∩ coUP by using ideas similar to those used for the proof of the upper bound for SC(∩, ×) (Theorem 10.50). Using prime decomposition, the problem reduces to an additive problem with respect to the exponents and since the prime decomposition is unique, we arrive at a UP ∩ coUP-problem.

Proposition 10.55. Let C be a {×}-circuit with $n \geq 0$ unassigned inputs where all gates are connected to the output gate. There are numbers $a, e_1, \ldots, e_n \in \mathbb{N}$ that can be computed in polynomial time such that for all $x_1, \ldots, x_n \in \mathbb{N}$,

$$\mathrm{I}(C(x_1, \ldots, x_n)) = a \cdot \prod_{i=1}^{n} x_i^{e_i}.$$

Proof. Observe that the statement is true for circuits where the output gate is an (assigned or unassigned) input gate. Furthermore, if the output gate is a ×-gate, the assertion holds by a straightforward induction argument. For the runtime, note that to obtain a, we have to compute a number of multiplications linear in the size of C. □

Theorem 10.56. SC(×) ∈ UP ∩ coUP.

Proof. Let C be a {×}-circuit with unassigned inputs $u_1, \ldots, u_n$ and let $b \geq 0$. We describe how to decide whether $(C, b) \in$ SC(×). We can assume that all gates in C are connected to the output gate. By Proposition 10.55, we can compute $a, e_1, \ldots, e_n \in \mathbb{N}$ such that for all $x_1, \ldots, x_n \in \mathbb{N}$,

$$\mathrm{I}(C(x_1, \ldots, x_n)) = a \cdot \prod_{i=1}^{n} x_i^{e_i}.$$

Hence, it suffices to find $x_1, \ldots, x_n \in \mathbb{N}$ such that $a \cdot x_1^{e_1} \cdots x_n^{e_n} = b$. The special cases $b = 0$ and b not being a multiple of a can be handled in polynomial time. Furthermore,

by replacing b by $\frac{b}{a}$, we can assume that $b \geq 1$ and $a = 1$. Suppose we know that $b = p_1^{d_1} \cdots p_m^{d_m}$ is the prime decomposition of b. Then the following statements are equivalent:

1. There are $x_1, \ldots, x_n \in \mathbb{N}$ such that $b = \prod_{i=1}^{n} x_i^{e_i}$.
2. For all $1 \leq j \leq m$ there are $y_{j,1}, \ldots, y_{j,n} \in \mathbb{N}$ such that $d_j = \sum_{i=1}^{n} y_{j,i} \cdot e_i$.

Algorithm 10.3 first guesses the prime decomposition of b nondeterministically and then decides whether the latter of the above statements is true and thus solves $\mathrm{SC}(\times)$.

Algorithm 10.3: `sc_times`$(b, e_1, \ldots, e_n)$

`Input` : Numbers $b, e_1, \ldots, e_n$

1 `guess numbers` $1 \leq m \leq \log b$, $2 \leq p_1 < \cdots < p_m \leq b$ `and` $1 \leq d_1, \ldots, d_m \leq \log b$;
2 `if at least one of the` p_j `is not prime then reject;`
3 `if` $b \neq p_1^{d_1} \cdots p_m^{d_m}$ `then reject;`
4 `if for all` $j \in \{1, \ldots, m\}$ `there are` $y_{j,1}, \ldots, y_{j,n} \in \mathbb{N}$ `such that` $\sum_{i=1}^{n} e_i y_{j,i} = d_j$ `then accept else reject`

Step 2 is possible in polynomial time by the algorithm by Agrawal, Kayal, and Saxena [AKS04]. Step 4 can also be carried out in polynomial time since the lengths of all d_j are logarithmic in the input length and so are the lengths of the $y_{j,i}$ (if they exist). Hence the $y_{j,i}$ can be found in nondeterministic logarithmic space and thus in polynomial time.

Since every number has a unique prime factorization, there exists exactly one path on which the algorithm reaches step 4. This shows $\mathrm{SC}(\times) \in \mathrm{UP}$. If we exchange `accept` and `reject` in step 4, then we arrive at an algorithm witnessing $\overline{\mathrm{SC}(\times)} \in \mathrm{UP}$. This completes the proof. □

We now show the NP-hardness of $\mathrm{SC}(^{-}, \cup, \cap)$ by reducing 3Sat to $\mathrm{SC}(^{-}, \cup, \cap)$. Here we utilize the natural correspondence between $\{^{-}, \cup, \cap\}$ and $\{\neg, \vee, \wedge\}$.

Theorem 10.57. $\mathrm{SC}(^{-}, \cup, \cap)$ is $\leq_{\mathrm{m}}^{\log}$-complete for NP.

Proof. By Corollary 10.46, $\mathrm{SC}(^{-}, \cup, \cap) \in \mathrm{NP}$. Moreover, $3\mathrm{Sat} \leq_{\mathrm{m}}^{\log} \mathrm{SC}(^{-}, \cup, \cap)$ by translating the Boolean operations $\neg, \vee, \wedge$ into the set operations $^{-}, \cup, \cap$ and by asking whether the resulting circuit can produce 1 (i. e., $A \subseteq \mathbb{N}$ is interpreted as *true* if and only if $1 \in A$). □

10.5 Conclusions

The results of this chapter are summarized in Table 10.1, which also contains the $\mathrm{MC}(\mathcal{O})$-problem for comparison. For the equivalence problems, we do not have matching lower and upper bound for any set of operations $\mathcal{O}$ where $\{\cup, +, \times\} \subseteq \mathcal{O}$. Note that due to the connections to Goldbach's conjecture, it is likely that the most general problem is undecidable. For the problems $\mathrm{EC}(\cap, +, \times)$ and $\mathrm{EC}(+, \times)$ it is unclear if randomization is needed or not (of course this also depends on whether randomization is helpful in general). So the open problems for $\mathrm{MC}(^{-}, \cup, \cap, +, \times)$ and $\mathrm{MC}(\cap, +, \times)$ from the article

$\mathcal{O}$	EC($\mathcal{O}$) Lower/Upper Bound	SC($\mathcal{O}$) Lower/Upper Bound	MC($\mathcal{O}$) Lower/Upper Bound
$^{-}\cup\cap+\times$	NEXP (10.16) $\quad$ Δ_2 (10.39)	Σ_1 (10.47) $\quad$ Σ_2 (10.43)	NEXP (10.16) $\quad$ Δ_2 (10.39)
$^{-}\cup\cap+$	PSPACE (10.16) $\quad$ (10.14)	PSPACE (10.43) $\quad$ (10.46)	PSPACE
$^{-}\cup\cap\ \times$	PSPACE (10.16) $\quad$ (10.41)	Σ_2 (10.43) $\quad$ PSPACE (10.43)	PSPACE
$^{-}\cup\cap$	P (10.16) $\quad$ (10.18)	NP (10.57) $\quad$ (10.46)	P
$\cup\cap+\times$	NEXP (10.16) $\quad$ coNEXP$^{\mathrm{NP}}$ (10.41)	Σ_1 (10.47) $\quad$ Σ_2 (10.43)	NEXP
$\cup\cap+$	PSPACE (10.16) $\quad$ (10.14)	PSPACE (10.43) $\quad$ (10.46)	PSPACE
$\cup\cap\ \times$	PSPACE (10.16) $\quad$ (10.41)	PSPACE (10.43) $\quad$ NEXP (10.53)	PSPACE
$\cup\cap$	P (10.16) $\quad$ (10.18)	P (10.43) $\quad$ (10.43)	P
$\cup\ +\times$	NEXP (10.42) $\quad$ coNEXP$^{\mathrm{NP}}$ (10.41)	PSPACE (10.43) $\quad$ (10.46)	PSPACE
$\cup\ +$	Π_2^{P} (10.36) $\quad$ (10.14)	NP (10.54) $\quad$ (10.46)	NP
$\cup\ \ \times$	Π_2^{P} (10.30) $\quad$ (10.36)	NP (10.43) $\quad$ (10.46)	NP
$\cup$	NL (10.16) $\quad$ (10.18)	NL (10.43) $\quad$ (10.43)	NL
$\cap+\times$	P (10.16) $\quad$ BPP (10.26)	Σ_1 (10.47) $\quad$ Σ_2 (10.43)	P (10.16) $\quad$ coRP (10.26)
$\cap+$	coC$_=$L(2) (10.24) $\quad$ (10.24)	NP (10.54) $\quad$ (10.46)	C$_=$L
$\cap\ \ \times$	coC$_=$L(2) (10.25) $\quad$ P (10.18)	NP (10.52) $\quad$ (10.50)	C$_=$L (10.25) $\quad$ P (10.18)
$\cap$	NL (10.16) $\quad$ (10.18)	NL (10.43) $\quad$ (10.43)	NL
$+\times$	P (10.16) $\quad$ coRP (10.10)	NP (10.54) $\quad$ (10.46)	P
$+$	C$_=$L (10.10) $\quad$ (10.10)	NP (10.54) $\quad$ (10.46)	C$_=$L
$\times$	C$_=$L (10.25) $\quad$ P (10.18)	NL (10.43) $\quad$ UP $\cap$ coUP (10.56)	NL

Table 10.1: Upper and lower bounds for EC($\mathcal{O}$), SC($\mathcal{O}$) and MC($\mathcal{O}$). All lower bounds are with respect to $\leq_{\mathrm{m}}^{\log}$-reductions and the numbers refer to where the result was obtained. If the upper and lower bounds coincide, only one is shown. The PSPACE-completeness of MC($\cup, +, \times$) was shown by Yang [Yan00]. All other unnumbered results on MC($\mathcal{O}$) are due to McKenzie and Wagner [MW07] who also showed the C$_=$L-completeness of EC($+$). EC($+, \times$) $\in$ coRP was shown by Schönhage [Sch79]. Note that the results from section 9.3 imply that MC($^{-}, \cup, \cap, +$) $\in$ DTIME($2^n\, n^4$) $\subseteq$ E.

by McKenzie and Wagner [MW07] transfer to the respective equivalence problems. For satisfiability problems, the main open question is whether $\mathrm{SC}(^{-}, \cup, \cap, \times)$ is decidable. In the absence of +-gates, we cannot express general Diophantine equations, which indicates the difficulty of proving undecidability. On the other hand, we do not know any decidable upper bound for this problem, since here the complementation-gates make it difficult to find a bound for the input gates. As the example in Figure 10.1(a) on page 173 shows, the set of all primes can be expressed by such circuits. A further open question is to find a better lower bound for the satisfiability problem for $\{\times\}$-circuits. We prove this problem to be in $\mathrm{UP} \cap \mathrm{coUP}$. Membership in P seems to be difficult, since $\mathrm{SC}(\times)$ comprises the following factoring-like problem: Is the factorization of a given number n of a certain form, for instance $n = x^3 \cdot y^5 \cdot z^2$? However, proving $\mathrm{SC}(\times)$ to be hard for factorization is still open.

Bibliography

[AAN82] V. Aggarwal, Y. P. Aneja, and K. P. K. Nair. Minimal spanning tree subject to a side constraint. *Computers & Operations Research*, 9(4):287 – 296, 1982.

[AB09] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.

[ABG04] E. Angel, E. Bampis, and L. Gourvès. Approximating the Pareto curve with local search for the bicriteria TSP(1, 2) problem. *Theoretical Computer Science*, 310(1-3):135–146, 2004.

[ABGM05] E. Angel, E. Bampis, L. Gourvès, and J. Monnot. (Non-)approximability for the multi-criteria TSP(1,2). In *Fundamentals of Computation Theory*, volume 3623 of *Lecture Notes in Computer Science*, pages 329–340. Springer Berlin / Heidelberg, 2005.

[ÀBJ95] C. Àlvarez, J. L. Balcázar, and B. Jenner. Adaptive logspace reducibility and parallel time. *Mathematical Systems Theory*, 28(2):117–140, 1995.

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

[AKS11] H.-C. An, R. Kleinberg, and D. B. Shmoys. Improving christofides' algorithm for the s-t path TSP, 2011.

[AL97] S. Arora and C. Lund. Hardness of approximations. In D. Hochbaum, editor, *Approximation Algorithms for NP-hard Problems*. PWS Publishing Company, Boston, 1997.

[ALM+92] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the intractability of approximation problems. In *Proceedings 33rd Symposium on the Foundations of Computer Science*, pages 14–23. IEEE Computer Society Press, 1992.

[AO96] E. Allender and M. Ogihara. Relationships among PL, #L, and the determinant. *RAIRO – Theoretical Informatics and Applications*, 30:1–21, 1996.

[APMS+99] G. Ausiello, M. Protasi, A. Marchetti-Spaccamela, G. Gambosi, P. Crescenzi, and V. Kann. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999.

[Aro98] S. Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *Journal of the ACM*, 45(5):753–782, 1998.

[Bak74] K. R. Baker. *Introduction to Sequencing and Scheduling.* Wiley, New York, 1974.

[Bal89] J. L. Balcázar. Self-reducibility structures and solutions of NP problems. *Revista Matematica de la Universidad Complutense de Madrid*, 2(2-3):175–184, 1989.

[BBFG91] R. Beigel, M. Bellare, J. Feigenbaum, and S. Goldwasser. Languages that are easier than their proofs. In *IEEE Symposium on Foundations of Computer Science*, pages 19–28, 1991.

[BCE+98] P. Beame, S. A. Cook, J. Edmonds, R. Impagliazzo, and T. Pitassi. The relative complexity of NP search problems. *Journal of Computer and System Sciences*, 57(1):3–19, 1998.

[BD76] A. B. Borodin and A. J. Demers. Some comments on functional self-reducibility and the NP hierarchy. Technical Report TR76-284, Cornell University, Department of Computer Science, 1976.

[BF81] J. Beck and T. Fiala. "Integer-making" theorems. *Discrete Applied Mathematics*, 3(1):1–8, 1981.

[BGS75] T. P. Baker, J. Gill, and R. Solovay. Relativizatons of the P =? NP question. *SIAM Journal on Computing*, 4(4):431–442, 1975.

[BKV03] M. Bezem, J.W. Klop, and R. Vrijer. *Term Rewriting Systems.* Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2003.

[BLS84] R. V. Book, T. Long, and A. L. Selman. Quantitative relativizations of complexity classes. *SIAM Journal on Computing*, 13:461–487, 1984.

[BM05] M. Bläser and B. Manthey. Approximating maximum weight cycle covers in directed graphs with weights zero and one. *Algorithmica*, 42(2):121–139, 2005.

[Bre07] H. Breunig. The complexity of membership problems for circuits over sets of positive numbers. In *Proceedings International Conference on Fundamentals of Computation Theory*, volume 4639 of *Lecture Notes in Computer Science*, pages 125–136. Springer-Verlag, 2007.

[BS85] J. L. Balcázar and U. Schöning. Bi-immune sets for complexity classes. *Mathematical Systems Theory*, 18(1):1–10, 1985.

[BS96] E. Bach and J. Shallit. *Algorithmic Number Theory*, volume I: Efficient Algorithms of *Foundations of Computing.* The MIT Press, Cambridge, MA, 1996.

[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, PA, 1976.

[CW90] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990.

[Doe05] B. Doerr. Integral approximation. Christian-Albrechts-Universität zu Kiel, 2005. Habilitation thesis.

[DP09] I. Düntsch and I. Pratt-Hartmann. Complex algebras of arithmetic. *Fundamenta Informaticae*, 97(4):347–367, 2009.

[DPR61] M. Davis, H. Putnam, and J. Robinson. The decision problem for exponential Diophantine equations. *Annals of Mathematics*, 74(2):425–436, 1961.

[DS03] B. Doerr and A. Srivastav. Multicolour discrepancies. *Combinatorics, Probability & Computing*, 12(4):365–399, 2003.

[EG00] M. Ehrgott and X. Gandibleux. A survey and annotated bibliography of multiobjective combinatorial optimization. *OR Spectrum*, 22(4):425–460, 2000.

[EG02] M. Ehrgott and X. Gandibleux, editors. *Multiple Criteria Optimization: State of the Art Annotated Bibliographic Survey*, volume 52 of *Kluwer's International Series in Operations Research and Management Science*. Kluwer Academic Publishers, 2002.

[Ehr00] M. Ehrgott. Approximation algorithms for combinatorial multicriteria optimization problems. *International Transactions in Operational Research*, 7:5–31, 2000.

[Ehr05] M. Ehrgott. *Multicriteria Optimization*. Springer Verlag, 2005.

[FGH+99] S. Fenner, F. Green, S. Homer, A. L. Selman, T. Thierauf, and H. Vollmer. Complements of multivalued functions. *Chicago Journal of Theoretical Computer Science*, 1999. Article 3 of volume 1999.

[FGL+12] K. Fleszar, C. Glaßer, F. Lipp, C. Reitwießner, and M. Witek. Structural complexity of multiobjective NP search problems. In *Proceedings 10th Latin American Theoretical Informatics Symposium (LATIN)*, 2012. to appear.

[FHOS97] S. Fenner, S. Homer, M. Ogihara, and A. L. Selman. Oracles that compute values. *SIAM Journal on Computing*, 26:1043–1065, 1997.

[FNW79] M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey. An analysis of approximations for finding a maximum weight Hamiltonian circuit. *Operations Research*, 27(4):799–809, 1979.

[FP74] M. J. Fischer and M. S. Paterson. String-matching and other products. Technical Report MIT-LCS-TM-041, Massachusetts Institute of Technology, 1974.

[FS74] M. J. Fischer and L. J. Stockmeyer. Fast on-line integer multiplication. *Journal of Computer and System Sciences*, 9(3), 1974.

[Für09] M. Fürer. Faster integer multiplication. *SIAM Journal on Computing*, 39(3):979–1005, 2009.

[GH03] A. Große and H. Hempel. On functions and relations. In *DMTCS*, pages 181–192, 2003.

[GHR+10] C. Glaßer, K. Herr, C. Reitwießner, S. D. Travers, and M. Waldherr. Equivalence problems for circuits over sets of natural numbers. *Theory of Computing Systems*, 46(1):80–103, 2010.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[GLLK79] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[GLS88] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, 1988.

[GR62] S. Ginsburg and H. G. Rice. Two families of languages related to ALGOL. *Journal of the ACM*, 9:350–371, 1962.

[Gre65] S. A. Greibach. A new normal-form theorem for context-free phrase structure grammars. *Journal of the ACM*, 12:42–52, 1965.

[GRS08] C. Glaßer, C. Reitwießner, and H. Schmitz. Multiobjective disk cover admits a PTAS. In *Proceedings of 19th International Symposium on Algorithms and Computation*, volume 5369 of *Lecture Notes in Computer Science*, pages 40–51. Springer Verlag, 2008.

[GRSW10] C. Glaßer, C. Reitwießner, H. Schmitz, and M. Witek. Approximability and hardness in multi-objective optimization. In *Proceedings Computability in Europe (CiE)*, volume 6158 of *Lecture Notes in Computer Science*. Springer Verlag, 2010.

[GRTW10] C. Glaßer, C. Reitwießner, S. D. Travers, and M. Waldherr. Satisfiability of algebraic circuits over sets of natural numbers. *Discrete Applied Mathematics*, 158(13):1394–1403, 2010.

[GRW09] C. Glaßer, C. Reitwießner, and M. Witek. Improved and generalized approximations for two-objective traveling salesman. *Electronic Colloquium on Computational Complexity (ECCC)*, 16:76, 2009.

[GRW11] C. Glaßer, C. Reitwießner, and M. Witek. Applications of discrepancy theory in multiobjective approximation. In *Proceedings Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 13 of *LIPIcs*, pages 55–65. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011.

[GSS92] S. A. Greibach, W. Shi, and S. Simonson. Single tree grammars. In *Theoretical Studies in Computer Science*, pages 73–99, 1992.

[GW86] A. Gupta and A. Warburton. Approximation methods for multiple criteria traveling salesman problems, towards interactive and intelligent decision support systems. In *Proceedings of 7th International Conference on Multiple Criteria Decision Making*, pages 211–217. Springer, 1986.

[Han79] P. Hansen. Bicriterion path problems. In *Proceedings of the 3rd Conference on Multiple Criteria Decision Making Theory and Application*, volume 177 of *Lecture Notes in Economics and Mathematical Systems*, pages 109–127. Springer Verlag, 1979.

[HHN+95] L. A. Hemaspaandra, A. Hoene, A. V. Naik, M. Ogihara, A. L. Selman, T. Thierauf, and J. Wang. Nondeterministically selective sets. *International Journal of Foundations of Computer Science*, 6(4):403–416, 1995.

[HNOS96] L. Hemaspaandra, A. Naik, M. Ogihara, and A. L. Selman. Computing solutions uniquely collapses the polynomial hierarchy. *SIAM Journal on Computing*, 25:697–708, 1996.

[Hoo91] J. A. Hoogeveen. Analysis of Christofides' heuristic: Some paths are more difficult than cycles. *Operations Research Letters*, 10:291–295, 1991.

[Hoo92] J. A. Hoogeveen. *Single-Machine Bicriteria Scheduling*. PhD thesis, Technical University of Eindhoven, 1992.

[HvdV90] J. A. Hoogeveen and S. L. van de Velde. Polynomial-time algorithms for single-machine multicriteria scheduling. Technical Report BS-R9008, Centre for Mathematics and Computer Science, Amsterdam, 1990.

[HW00] H. Hempel and G. Wechsung. The operators min and max on the polynomial hierarchy. *International Journal of Foundations of Computer Science*, 11(2):315–342, 2000.

[HZ80] G. Y. Handler and I. Zang. A dual algorithm for the constrained shortest path problem. *Networks*, 10(4):293–309, 1980.

[Imm88] N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17:935–938, 1988.

[Jeż08] A. Jeż. Conjunctive grammars generate non-regular unary languages. *International Journal of Foundations of Computer Science*, 19(3):597–615, 2008.

[JO08] A. Jeż and A. Okhotin. On the computational completeness of equations over sets of natural numbers. In *Automata, Languages and Programming*, volume 5126 of *Lecture Notes in Computer Science*, pages 63–74. Springer Berlin / Heidelberg, 2008.

[JO10] A. Jeż and A. Okhotin. Conjunctive grammars over a unary alphabet: Undecidability and unbounded growth. *Theory Computing Systems*, 46(1):27–58, 2010.

[JO11a] A. Jeż and A. Okhotin. Complexity of equations over sets of natural numbers. *Theory of Computing Systems*, 48:319–342, 2011. 10.1007/s00224-009-9246-y.

[JO11b] A. Jeż and A. Okhotin. One-nonterminal conjunctive grammars over a unary alphabet. *Theory of Computing Systems*, 49:319–342, 2011. 10.1007/s00224-011-9319-6.

[JP85] D. S. Johnson and C. H. Papadimitriou. Performance guarantees for heuristics. In E. L. Lawler, Jan Karel Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, editors, *The Traveling Salesman Problem. A Guided Tour of Combinatorial Optimization*, chapter 5, pages 145–180. Wiley, 1985.

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

[KH66] A. J. Korenjak and J. E. Hopcroft. Simple deterministic languages. *IEEE Symposium on Switching and Automata Theory*, pages 36–46, 1966.

[KLSS05] H. Kaplan, M. Lewenstein, N. Shafrir, and M. Sviridenko. Approximation algorithms for asymmetric TSP by decomposing directed regular multigraphs. *Journal of the ACM*, 52(4):602–626, 2005.

[KNR09] V. Kountouriotis, C. Nomikos, and P. Rondogiannis. Well-founded semantics for Boolean grammars. *Information and Computation*, 207(9):945 – 967, 2009.

[Ko82] K. Ko. Some observations on the probabilistic algorithms and NP-hard problems. *Information Processing Letters*, 14(1):39–43, 1982.

[Kre88] M. W. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36:490–509, 1988.

[Lau83] C. Lautemann. BPP and the polynomial hierarchy. *Information Processing Letters*, 17:215–217, 1983.

[Llo78] N. G. Lloyd. *Degree Theory*. Cambridge University Press, Cambridge, England, 1978.

[MA78] K. L. Manders and L. M. Adleman. NP-complete decision problems for binary quadratics. *Journal of Computer and System Sciences*, 16(2):168–184, 1978.

[Man08] B. Manthey. On approximating restricted cycle covers. *SIAM Journal on Computing*, 38(1):181–206, 2008.

[Man09] B. Manthey. On approximating multi-criteria TSP. In *Proceedings of 26th Annual Symposium on Theoretical Aspects of Computer Science*, volume 09001 of *Dagstuhl Seminar Proceedings*, pages 637–648. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2009.

[Man11] B. Manthey. Deterministic algorithms for multi-criteria TSP. In *Proceedings of the International Conference on Theory and Applications of Models of Computation*, volume 6648 of *Lecture Notes in Computer Science*, pages 264–275. Springer Verlag, 2011.

[Mat70] Y. V. Matiyasevich. Enumerable sets are Diophantine. *Doklady Akad. Nauk SSSR*, 191:279–282, 1970. Translation in Soviet Math. Doklady, 11:354–357, 1970.

[MR09] B. Manthey and L. S. Ram. Approximation algorithms for multi-criteria traveling salesman problems. *Algorithmica*, 53(1):69–88, 2009.

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

[MS11] T. Mömke and O. Svensson. Approximating graphic TSP by matchings. In *FOCS '11: Proceedings of the 52nd Annual Symposium on Foundations of Computer Science*, pages 560–569. IEEE Computer Society, 2011.

[MW07] P. McKenzie and K. W. Wagner. The complexity of membership problems for circuits over sets of natural numbers. *Computational Complexity*, 16(3):211–244, 2007.

[OGSS11] S. Oveis Gharan, A. Saberi, and M. Singh. A randomized rounding approach to the traveling salesman problem. In *FOCS '11: Proceedings of the 52nd Annual Symposium on Foundations of Computer Science*, pages 550–559. IEEE Computer Society, 2011.

[Okh01] A. Okhotin. Conjunctive grammars. *Journal of Automata, Languages and Combinatorics*, 6(4):519–535, 2001.

[Okh04] A. Okhotin. Boolean grammars. *Information and Computation*, 184(1):19–48, 2004.

[Okh07] A. Okhotin. Recursive descent parsing for Boolean grammars. *Acta Informatica*, 44(3-4):167–189, 2007.

[Okh08] A. Okhotin. Unambiguous Boolean grammars. *Information and Computation*, 206:1234–1247, 2008.

[Okh10] A. Okhotin. Fast parsing for Boolean grammars: a generalization of Valiant's algorithm". In *Proceedings of DLT 2010*, volume 6224 of *Lecture Notes in Computer Science*, pages 340–351. Springer Verlag, 2010.

[Okh11] A. Okhotin. Conjunctive and Boolean grammars: the true general case of the context-free grammars. 2011. to appear.

[OR10] A. Okhotin and C. Reitwießner. Conjunctive grammars with restricted disjunction. *Theoretical Computer Science*, 411(26-28):2559–2571, 2010.

[OR11] A. Okhotin and C. Reitwießner. Parsing Unary Boolean Grammars Using Online Convolution. In Christian Glasser, Jean-Eric Pin, Nicole Schweikardt, Victor Selivanov, and Wolfgang Thomas, editors, *Advances and Applications of Automata on Words and Trees*, number 10501 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2011. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

[Pap94] C. M. Papadimitriou. *Computational complexity*. Addison-Wesley, Reading, Massachusetts, 1994.

[PD09] I. Pratt-Hartmann and I. Düntsch. Functions definable by arithmetic circuits. In *CiE, Computability in Europe*, volume 5635 of *Lecture Notes in Computer Science*, pages 409–418. Springer, 2009.

[PMM09] K. Paluch, M. Mucha, and A. Madry. A 7/9 - approximation algorithm for the maximum traveling salesman problem. In I. Dinur, K. Jansen, J. Naor, and J. Rolim, editors, *Proceedings of APPROX/RANDOM*, volume 5687 of *Lecture Notes in Computer Science*, pages 298–311. Springer Berlin / Heidelberg, 2009.

[PV06] C. Papadimitriou and S. Vempala. On the approximability of the traveling salesman problem. *Combinatorica*, 26(1):101–120, 2006.

[PY93] C. Papadimitriou and M. Yannakakis. The traveling salesman problem with distances one and two. *Mathematics of Operations Research*, 18(1):1–11, 1993.

[PY00] C. H. Papadimitriou and M. Yannakakis. On the approximability of trade-offs and optimal access of web sources. In *FOCS '00: Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, pages 86–95, Washington, DC, USA, 2000. IEEE Computer Society.

[Rei07] C. Reitwießner. Formal languages defined by recurrent circuits. Universität Würzburg, 2007. Diploma thesis.

[RG96] R. Ravi and M. X. Goemans. The constrained minimum spanning tree problem (extended abstract). In *Algorithm Theory - SWAT '96*, volume 1097 of *Lecture Notes in Computer Science*, pages 66–75. Springer-Verlag, 1996.

[Sch79] A. Schönhage. On the power of random access machines. In *ICALP*, pages 520–529, 1979.

[Sel92] A. L. Selman. A survey of one-way functions in complexity theory. *Mathematical Systems Theory*, 25:203–221, 1992.

[Sel94] A. L. Selman. A taxonomy on complexity classes of functions. *Journal of Computer and System Sciences*, 48:357–381, 1994.

[Sel96] A. L. Selman. Much ado about functions. In *Proceedings 11th Conference on Computational Complexity*, pages 198–212. IEEE Computer Society Press, 1996.

[Sip83] M. Sipser. A complexity theoretic approach to randomness. In *Proceedings of the 15th Symposium on Theory of Computing*, pages 330–335, 1983.

[SM73] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time. In *Proceedings 5th ACM Symposium on the Theory of Computing*, pages 1–9. ACM Press, 1973.

[Sze88] R. Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26:279–284, 1988. 10.1007/BF00299636.

[Tra06] S. Travers. The complexity of membership problems for circuits over sets of integers. *Theoretical Computer Science*, 369(1):211–229, 2006.

[Val75] L. G. Valiant. General context-free recognition in less than cubic time. *Journal of Computer and System Sciences*, 10(2):308–314, 1975.

[Val76] L. G. Valiant. Relative complexity of checking and evaluating. *Information Processing Letters*, 5(1):20–23, 1976.

[Voi31] *Der Handlungsreisende, wie er sein soll, und was er zu thun hat, um Aufträge zu erhalten und eines glüklichen Erfolgs in seinen Geschäften gewiß zu sein. Von einem alten Commis Voyageur.* Bernhard Friedrich Voigt, Ilmenau, 1831.

[VY05] S. Vassilvitskii and M. Yannakakis. Efficiently computing succinct trade-off curves. *Theoretical Computer Science*, 348(2-3):334–356, 2005.

[Wag84] K. Wagner. The complexity of problems concerning graphs with regularities. In *Proceedings Mathematical Foundations of Computer Science*, volume 176 of *Lecture Notes in Computer Science*, pages 544–552. Springer-Verlag, 1984.

[Wec00] G. Wechsung. *Vorlesungen zur Komplexitätstheorie*, volume 32 of *TEUBNER-TEXTE zur Informatik*. B. G. Teubner, 2000.

[Wot73] D. Wotschke. The Boolean closures of deterministic and nondeterministic context-free languages. In W. Brauer, editor, *Gesellschaft für Informatik e.V., 3. Jahrestagung*, volume 1 of *LNCS*, pages 113–121, 1973.

[Yan00] K. Yang. Integer circuit evaluation is PSPACE-complete. In *IEEE Conference on Computational Complexity*, pages 204–213, 2000.

Notations

Index