Multiobjective Traveling Salesman Problems and Redundancy of Complete Sets

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

vorgelegt von

Maximilian Witek

aus Trier

Würzburg, 2014

Eingereicht am: 20.8.2014
bei der Fakultät für Mathematik und Informatik
1. Gutachter: Prof. Dr. Christian Glaßer
2. Gutachter: Prof. Dr. Pavan Aduri
Tag der mündlichen Prüfung: 4.2.2015

Acknowledgements

I am grateful for all the support I received during my studies and the completion of this thesis.

Most of all, I would like to thank my advisor Christian Glaßer for his continuous support and for sharing his profound understanding of theory and mathematics with me.

Special thanks go to Heinz Schmitz, who encouraged my interest in the theoretical aspects of computer science and introduced me to research at an early stage of my studies.

I owe my gratitude to Klaus Wagner and Alexander Wolff, who both gave me the opportunity to work at their chairs.

I want to thank Christian Reitwießner for the excellent and fruitful collaboration.

I further thank Alan L. Selman and Dung T. Nguyen for the joint work on autoreducibility and mitoticity. Moreover, I thank Pavan Aduri for agreeing to examine this thesis.

Finally, I want to thank my family and friends for their patience and support.

Abstract

The first part of this thesis deals with the approximability of the traveling salesman problem. This problem is defined on a complete graph with edge weights, and the task is to find a cycle of minimum or maximum weight that visits each vertex exactly once. We study the most important multiobjective variants of this problem. In the multiobjective case, the edge weights are vectors of natural numbers, and we approximate the Pareto set, which contains those solutions whose weights are optimal tradeoffs between the different objectives.

We first study the two-objective minimization version of the traveling salesman problem with metric weight functions. We obtain improved approximation algorithms and show that similar techniques lead to approximation algorithms for related Hamiltonian path problems. We demonstrate that our algorithms also apply to more general multigraph versions of the problem, and we develop arguments against a significant improvement of our results.

Furthermore, we investigate the k-objective maximization versions of the traveling salesman problem. We distinguish between directed and undirected graphs, and in both cases, we develop improved approximation algorithms by transferring single-objective algorithms to the multiobjective case. The single-objective algorithms compute a maximum weight cycle cover and remove the lightest edge per cycle. A trivial adaption of these algorithms to the multiobjective case fails, because multiobjective weights can be incomparable, and the meaning of the term “lightest edge” is unclear. We develop a general lemma that helps to deal with this difficulty. While our algorithms provide the currently best approximation ratio for the k-objective maximization versions, they have the disadvantage of being randomized. We show that with a very small deviation in the approximation ratio, we can obtain deterministic versions of our algorithms.

In the second part of this thesis, we study redundancy properties of complete sets. We call a set A autoreducible if membership information about an input instance x can be efficiently determined from the membership information of other instances $y \neq x$. If A can be split into two equivalent parts, then it is called weakly mitotic, and if the splitting is obtained by an efficiently decidable separator set, then it is called mitotic.

For a given reducibility notion and complexity class, we analyze how redundant its complete sets are. Previous research in this field concentrates on polynomial-time reducibility notions, often on many-one or Turing complete sets. Our main contribution is a systematic study of the redundancy of logspace complete sets for typical complexity classes. Depending on the computational strength of the complexity class and reducibility notion considered, we use different techniques to show redundancy. For small complexity classes such as NL and P, we apply self-reducibility to establish autoreducibility, while for large complexity classes such as PSPACE and EXP, which have enough computational power to simulate arbitrary logspace reductions, we use diagonalization to establish mitoticity. For intermediate classes such as NP and the levels of the polynomial-time hierarchy, we use local checkability to provide autoreducibility of complete sets.

In many cases, we obtain autoreducibility, while mitoticity is not known to hold. We finish our study by an adaption of a deterministic coin tossing technique to the logspace setting, which shows that in some cases, autoreducibility at least implies weak mitoticity.

Contents

Chapter 1

Introduction

A central subject of theoretical computer science is the study of complexity classes. Such classes categorize algorithmic problems according to the computational resources that are needed to solve them. Most prominently, the class P contains those problems that can be solved in efficient runtime, and the class NP contains those problems whose solutions can be efficiently verified. Today, there exists a large number of important problems that are known to belong to NP, but whose membership to P is yet unknown. For that reason, the question whether P equals NP has become one of the most important and popular open questions in mathematics.

The general subject of this thesis is the study of complete problems for NP and further complexity classes. Such problems are the most difficult problems that belong to a given complexity class, and they are closely related to the class itself. For instance, in order to show that P equals NP, it suffices to show that an NP-complete problem belongs to P. Many practical problems in NP turned out to be NP-complete, hence their study is a challenging and difficult task.

This thesis studies complete problems from two perspectives. In the first part, we study multiobjective optimization variants of the traveling salesman problem, a prominent example of an NP-complete problem. In the second part of this thesis, we look at complete problems in a very general way, and we analyze redundancy properties that hold for all complete problems of a given complexity class. Below, we give a more detailed introduction to these topics.

1.1 Multiobjective Traveling Salesman Problems

Suppose we are given a set of cities on a map, and we want to find the shortest roundtrip that visits every city exactly once. This problem is known as the minimum traveling salesman problem (MinTSP) and has become one of the most studied combinatorial optimization problems in mathematics until today. More generally, it is defined as a graph problem, where the input consists of a complete graph with edge weights, and we want to find a Hamiltonian cycle of minimum weight. This general problem has a large number of practical applications, such as vehicle routing, job sequencing, and computer wiring [LLKS85].

Despite its simple formulation, all efforts to find efficient algorithms that solve MinTSP so far have failed. Karp [Kar72] gave a mathematical explanation by showing the NP-hardness of MinTSP. This means that MinTSP is at least as difficult as all problems that are contained in the complexity class NP, such as scheduling, bin packing, partitioning, and many more practically relevant problems (see Garey and Johnson [GJ79] for a survey). Moreover, a fast algorithm for MinTSP would show that P equals NP and hence would yield fast algorithms for all of these problems. Today there exist many further arguments that indicate how difficult it is to show that P equals NP, hence we cannot expect to find an exact and fast solution algorithm for the general MinTSP, and we need other ways to cope with its NP-hardness.

Recall that in the original problem formulation we want to find a Hamiltonian cycle of minimum weight. If we relax the optimality condition and search for Hamiltonian cycles that have a small but not necessarily minimal weight, then finding a solution to our problem becomes easier. Such a relaxation of the original problem is captured by the notion of an approximation algorithm. For $\alpha \geq 1$, an algorithm is called α-approximation for MinTSP if it runs fast and always finds a Hamiltonian cycle that weighs at most α times the minimal possible weight. If MinTSP has an α-approximation, then it is simply called α-approximable.

A large amount of research has been spent on the approximability of MinTSP. For instance, Sahni and Gonzalez [SG76] showed that there does not exist an α-approximation of MinTSP (where α is an arbitrary constant), unless P = NP, while Christofides [Chr76] showed that its restriction to metric problem instances is $3/2$-approximable. Interestingly, Christofides' algorithm is still the best known approximation algorithm for metric MinTSP.

In the first part of this thesis, we will study the approximability of the traveling salesman problem in the presence of multiple objectives. We will concentrate on the following k-objective problem variants, where $k \in \mathbb{N}$ denotes the number of objectives:

k-MinΔTSP: For a complete undirected graph with edge weights in $\mathbb{N}^k$ that satisfy the triangle inequality in each objective, find Hamiltonian cycles of minimum weight.

k-MaxSTSP: For a complete undirected graph with edge weights in $\mathbb{N}^k$, find Hamiltonian cycles of maximum weight.

k-MaxATSP: For a complete directed graph with edge weights in $\mathbb{N}^k$, find Hamiltonian cycles of maximum weight.

Observe that in the multiobjective setting, the weights of Hamiltonian cycles are vectors of natural numbers and hence can be incomparable, so there might not exist a unique optimal solution. The Pareto set as the set of all optimal tradeoffs captures the notion of optimality in this setting, and our goal is to compute sets of solutions that approximate the Pareto set.

The previously best known approximation algorithms for these problems are due to Manthey and Ram [MR09] and Manthey [Man12b, Man12a] (we refer to Chapter 3 for a summary of previous work). Our main contributions are as follows:

- In the case of k-MinΔTSP, we concentrate on two objectives. We work with Christofides-like algorithms and obtain a 2-approximation. We further use our technique to approximate similar Hamiltonian path problems, and we show that our algorithms actually solve more general multigraph problems that cover the metric problems as a special case.
- In the case of k-MaxATSP and k-MaxSTSP, we generalize known cycle cover based algorithms to the multiobjective case and again obtain improved approximation results.

While our minimization algorithms improve the previously best known approximation algorithms, they leave a significant gap to the approximation ratio that is obtained by Christofides' algorithm in the single-objective case. We give a partial explanation of this discrepancy by showing that significant improvements of our results yield improved algorithms for well-studied single-objective problems.

The maximization algorithms that we develop match the approximation ratio of their single-objective counterpart. In order to generalize these algorithms to the case of multiple objectives, we solve a very general problem: given a list of multidimensional weights and some $c \geq 1$, we can find a selection that roughly weighs a fraction of $1/c$ of the entire list in each objective. We demonstrate how to apply this result to approximate further multiobjective problems.

1.2 Redundancy of Complete Sets

In the second part of this thesis we shift our focus to general properties of complete sets. Consider, for instance, the following question:

- Are all NP-complete sets infinite?

It is quite often the case that such questions can easily be answered for all *known* complete problems, while an answer that holds for *all* complete problems has important consequences. In the above case, it is easy to see that all problems that are known to be NP-complete (such as the traveling salesman problem) are infinite. However, if *all* NP-complete problems are infinite, then finite sets in P are not NP-complete and hence $\mathrm{P} \neq \mathrm{NP}$, and if some NP-complete problem is finite, then all problems in NP are reducible to a finite set and hence polynomial-time decidable. So the above question is just another formulation of the question whether P equals NP.

Instead of infiniteness, we will consider redundancy properties and ask similar questions for complete sets for different complexity classes and with respect to different reducibility notions. So suppose we have a reducibility notion $\leq$ and a complexity class $\mathcal{C}$. Our general question in the second part of this thesis will be as follows:

- Are all $\leq$-complete sets for $\mathcal{C}$ redundant?

A strong form of redundancy is the paddability property. This notion goes back to the work of Berman and Hartmanis [BH77]. Inspired by the similarities between P and REC (resp., NP and RE), they tried to show that all NP-complete sets are polynomial-time isomorphic to each other. They observed that a set is polynomial-time isomorphic to SAT if and only if it is paddable, which means that in polynomial time, one can encode and decode arbitrary information into its instances without changing their membership status to the set. Observe that for arbitrary elements, the membership information to a paddable set is redundantly stored in the membership information of a large number of different elements, so we can think of paddability as a strong form of redundancy. For all *known* NP-complete problems, Berman and Hartmanis could show paddability, and they conjectured that paddability holds for *all* NP-complete problems, and hence that all NP-complete problems are polynomial-time isomorphic to each other. Observe that a positive answer to their conjecture would imply $\mathrm{P} \neq \mathrm{NP}$, because SAT is not isomorphic to finite sets. The conjecture is still unsolved, and we study weaker forms of redundancy in order to obtain partial progress on these difficult questions.

The properties of autoreducibility and mitoticity have their origins in recursion theory. Trakhtenbrot [Tra70] called a set A autoreducible if there exists an algorithm that determines the membership of x with additional access to the membership information of arbitrary elements $y \neq x$ to A, and Ladner [Lad73] introduced mitoticity for sets that are a disjoint union of two sets in the same degree. It is known that in general, mitoticity implies autoreducibility. Ladner showed that for recursively enumerable sets, also the reverse implication holds, so here, autoreducibility and mitoticity are equivalent.

Ambos-Spies [AS84] studied these concepts in the complexity-theoretic setting. With respect to polynomial-time many-one reductions, he called a set A autoreducible if A reduces to itself via a reduction function f that never maps to its own input, and he called A weakly mitotic if A can be split by a separator set S into two parts $A \cap S$ and $A \cap \overline{S}$ such that A and the two parts are equivalent to each other. If additionally $S \in \mathrm{P}$, then A is simply called mitotic. Note that for further reducibility notions, one can define autoreducibility and mitoticity analogously.

The concepts of autoreducibility and mitoticity can be interpreted as weaker forms of redundancy. If a set is autoreducible, then membership information about its elements can be derived locally from that of other elements. Moreover, the membership information of elements of a mitotic set is redundantly stored in two parts of the set. Consequently, we will think of autoreducibility and mitoticity as local and global forms of redundancy. Note that mitoticity generally implies autoreducibility, while the converse does not necessarily hold. So for a reducibility notion $\leq$ and a complexity class $\mathcal{C}$, our redundancy questions will be as follows:

Are all $\leq$-complete sets for $\mathcal{C}$ $\leq$-autoreducible or even $\leq$-mitotic?

A significant amount of research has been spent on the study of this question for polynomial-time reducibility notions. For instance, the following results are known for non-trivial sets:

- All $\leq_{\mathrm{m}}^{\mathrm{p}}$-complete sets for NP are $\leq_{\mathrm{m}}^{\mathrm{p}}$-mitotic [GPSZ08].
- All $\leq_{\mathrm{dtt}}^{\mathrm{p}}$-complete sets for PSPACE are $\leq_{\mathrm{dtt}}^{\mathrm{p}}$-autoreducible [GOP+07].
- There exists a $\leq_{\mathrm{btt}}^{\mathrm{p}}$-complete set for EXP that is not $\leq_{\mathrm{btt}}^{\mathrm{p}}$-autoreducible [BFvMT00].

These few results already lead to very interesting situations. The first two results show that even though complete sets are the most difficult sets of a complexity class, they contain some kind of regularity, while this might not hold for all sets of the class. Moreover, extending the first result to all $\leq_{\mathrm{btt}}^{\mathrm{p}}$-complete sets for NP is very difficult, because, by the third result, there are $\leq_{\mathrm{btt}}^{\mathrm{p}}$-complete sets for EXP that are not $\leq_{\mathrm{btt}}^{\mathrm{p}}$-autoreducible, and hence we would have separated NP and EXP. On the other hand, showing the third result for NP would separate P and NP, because all sets in P are trivially $\leq_{\mathrm{btt}}^{\mathrm{p}}$-autoreducible.

A lot of further redundancy results are known for various reducibility notions and complexity classes (we refer to Chapter 7 for a detailed summary). However, most research in this field concentrates on polynomial-time reducibility notions. While we obtain some progress on these questions, our main contribution is a systematic study of autoreducibility and mitoticity of sets that are complete with respect to logspace reducibility notions. Using logspace reductions enables us to further look into small classes such as NL and P, but at the same time it restricts the techniques we can use. Consequently, we will use different techniques to establish autoreducibility or even mitoticity that depend on the particular context:

- For small classes such as NL or P, we will use a tree-traversal technique that is based on the concept of self-reducibility to establish autoreducibility properties.
- For large classes such as PSPACE or EXP, we will apply diagonalization techniques to enforce mitoticity properties.
- For intermediate classes such as NP and the levels of the polynomial-time hierarchy, we will use techniques that locally check objects such as computational transcripts or Boolean formulas to establish autoreducibility properties.

In many cases, we obtain that complete sets are autoreducible, and the question remains open whether these sets are mitotic. We conclude our study with a very general result, which shows that in some cases, autoreducibility at least implies weak mitoticity. Improvements to mitoticity remain a challenging task for future work.

1.3 Outline

Chapter 2 contains notations and mathematical foundations that are relevant for the entire thesis. The remainder of the thesis is split into two parts. In Chapters 3 to 6, we study the approximability of multiobjective traveling salesman problems. In Chapters 7 to 11, we study autoreducibility and mitoticity properties of complete sets. Chapters 3 and 7 are introductory chapters to each of the two parts of the thesis. At the end of Chapter 3 we give a summary of the first part, and at the end of Chapter 7 we give a summary of the second part.

Below we give a more detailed description of the contents of each chapter.

Chapter 2

This chapter serves as a reference for notations and definitions that we use throughout this thesis. It includes the basic mathematical notations, the graph-theoretic concepts, a notation for Boolean formulas and operators, and the complexity-theoretic foundations on which this thesis is based. The first part of this thesis deals with multiobjective traveling salesman problems, for which the graph-theoretic concepts are of particular importance. The complexity-theoretic concepts are important for the second part of the thesis that deals with redundancy properties of complete sets.

Chapter 3

In this chapter we give a general introduction to multiobjective optimization and traveling salesman problems. We use the notation that we developed in previous papers [GRSW10a, FGL$^+$12] to give a precise definition of multiobjective problems and their solution, and we show how to approximately solve such problems. We further define the relevant multiobjective traveling salesman problems and additional multiobjective graph problems that we consider in the first part of this thesis. In particular, we work with multiobjective versions of the minimum perfect matching problem, the minimum spanning tree problem, and the minimum shortest path problem. We include the important result by Papadimitriou and Yannakakis [PY00], who showed that these problems admit polynomial-time approximation schemes.

Chapter 4

We first study multiobjective versions of the minimum traveling salesman problem and related path problems. Since a constant-ratio approximation algorithm for the general single-objective minimum traveling salesman problem would imply P = NP, one often works on restrictions with metric distance functions. We argue that a straightforward generalization to multiple objectives (i.e., multiobjective distance functions that are metric in each component) is very restrictive, and a more general and practically relevant multiobjective version can be obtained by considering spanning walks in multigraphs. We show reductions that adapt the approximation results by Papadimitriou and Yannakakis [PY00] for multiobjective matching, shortest path and spanning tree problems to the corresponding multigraph problems. As the main result of this chapter, we give a deterministic and a randomized approximation algorithm for the two-objective minimum traveling salesman problem on multigraphs. Our deterministic algorithm is a modification of a tree-doubling technique, while our randomized algorithm is inspired by Christofides' algorithm [Chr76]. Moreover, we develop similar algorithms for the traveling salesman path problems on multigraphs. We finish the chapter with lower bound considerations, where we show that certain improvements of the multiobjective case would yield improvements of well-studied single-objective problems.

Chapter 5

We shift our focus to maximization variants of the multiobjective traveling salesman problem. We allow arbitrary multiobjective weights on the edges and distinguish between the problem version on directed and on undirected graphs. For both versions there exist basic approximation algorithms in the single-objective setting. These algorithms first compute a maximum weight cycle cover, then remove the lightest edge per cycle, and then connect the remaining paths to a Hamiltonian cycle. Since in directed and undirected graphs, each cycle consists of at least two and three edges, one obtains a $1/2$ and a $2/3$ approximation, respectively. For more than one objective, the situation is more complicated, because then the term “lightest edge” is not well-defined, and a straightforward adaption of the algorithm to multiple objectives fails. The main result in this chapter is a lemma that solves this problem in a very general way. We use results from discrepancy theory to show that for a matrix with c rows of multidimensional vectors, we can choose one vector per column such that the selection roughly weighs $1/c$ of the entire matrix in each dimension. Moreover, we can compute such a selection in polynomial time. These results provide us with an algorithm that chooses for each cycle of a cycle cover an edge that we can remove without losing too much overall weight. We are hence able to transfer the single-objective approximation algorithms to the multiobjective case, where we obtain a $1/2$ and a $2/3$ approximation, respectively. We note that these algorithms rely on the matching approximation by Papadimitriou and Yannakakis [PY00] and hence are randomized. Moreover, we show that our technique is not restricted to traveling salesman problems but rather solves a problem that generally occurs in multiobjective optimization. As another example, we use our lemma to generalize a basic approximation algorithm for the single-objective maximum satisfiability problem on Boolean formulas to multiple objectives.

Chapter 6

We finish the first part of this thesis by considering the deterministic approximation properties of the multiobjective maximum traveling salesman problems. Our results rely on necklace splitting results by Stromquist and Woodall [SW85], who showed how to split necklaces that consist of different materials with very few cuts into evenly balanced parts. Their result provides us with a vector balancing lemma that is similar to the result in the previous chapter, but which has more structure. Namely, given a matrix of c rows of multidimensional vectors, we can choose one vector per column such that the entire selection roughly weighs $1/c$ in each dimension. Moreover, this selection consists of very few row blocks that together form the selection. We show that this lemma enables us to combine different matchings in a convex way into a single matching that roughly has the same weight as a convex combination of the different matching weights. Together with an algorithmic version of Carathéodory’s theorem [Car11], we can find for arbitrary multidimensional weight vectors d a matching that almost weighs as much as d, if such a matching exists. By using standard reductions, this provides us with a PTAS-like approximation algorithm for cycle cover, and by applying an algorithm that is similar to the one we used in the previous chapter, we obtain deterministic $(1-\varepsilon)/2$ and $(2-\varepsilon)/3$ approximation algorithms for the multiobjective maximum traveling salesman problems for arbitrary $\varepsilon > 0$.

Chapter 7

We continue with a general introduction to the study of redundancy properties of complete sets. We first give a general introduction to this field. Next we define the concepts of autoreducibility, mitoticity, and weak mitoticity. We summarize the known results for various reducibility notions and complexity classes, before we finish the chapter with some simple observations and properties that are helpful for the remainder of the second part of this thesis. We include an overview of the currently known autoreducibility, mitoticity, and weak mitoticity results of complete sets.

Chapter 8

We first show a technique to establish autoreducibility of complete sets that is based on self-reducibility. This property of sets was introduced by Balcázar [Bal90] and denotes a stronger form of autoreducibility, where for an arbitrary input x, the membership information to the self-reducible set can be obtained from the membership information of elements y that are strictly smaller than x. We show that if for some class there exists a complete set that is self-reducible, then its autoreducibility applies to all complete sets in that class. This is particularly useful for small classes such as NL and P, where complete, self-reducible sets are known to exist. We hence obtain autoreducibility of complete sets for NL and P with respect to various reducibility notions. Moreover, a modification of the self-reducibility technique shows autoreducibility with respect to truth-table reductions that only use fixed binary Boolean functions.

Chapter 9

We continue our investigation by applying local checkability to complete sets. Suppose, for instance, we have a complete set A for P and a Turing machine M that shows $A \in \mathrm{P}$. We consider the transcript (i.e., the sequence of configurations) of M on some input x. With an appropriate encoding, each transcript bit only depends on a constant number of previous bits. So if we are given a transcript, then we can locally check its consistency by a bitwise verification. Note that the transcript bits of a polynomial-time computation are computable in polynomial time and hence reducible to A. By reducing the transcript bits to the sets $A \cup \{x\}$ and $A - \{x\}$ we obtain two candidate transcripts without querying x. Using the local checkability technique we can verify the correct transcript and determine whether $x \in A$ holds or not.

With this technique we obtain new autoreducibility results for logspace bounded truth-table complete sets for P and the Δ_k^{P}-levels of the polynomial-time hierarchy. As a second application, we transfer redundancy results from the polynomial-time setting to the logspace setting by looking at transcripts of polynomial-time autoreductions. This is particularly useful for logspace-complete sets for NP or coNP, where so far, only polynomial-time mitoticity was known. We show that these sets are logspace Turing autoreducible. We conclude the chapter with an adaption of negative results by Buhrman et al. [BFvMT00] to the logspace setting. We obtain that improving some of the results shown in this chapter is difficult, as it would lead to new separations of complexity classes.

Chapter 10

We proceed by analyzing complete sets of classes that are sufficiently powerful to simulate logspace or polynomial-time reductions, such as PSPACE, EXP, and NEXP. For these sets we use diagonalization to prevent "unwanted cases" in the reductions. For instance, if we consider a complete set A for PSPACE, then we can define another complete set B that enforces every reduction to A on input x to query a value different from x, which enables us to show autoreducibility. In some cases, we can enforce the autoreduction to be length-squaring, and by standard techniques, this provides even mitoticity. A similar technique applies to complete sets for PSPACE and EXP with respect to various logspace or polynomial-time reducibility notions. For NEXP, the situation is more complicated, because we do not know whether NEXP is closed under complementation or not. Using diagonalization techniques we show that complete sets for NEXP with respect to various logspace and polynomial-time reducibility notions are autoreducible.

Chapter 11

We finish our study of redundancy of complete sets by showing that in some cases, logspace autoreducibility of complete sets implies weak logspace mitoticity. The difference to mitoticity is that we do not require the separator that splits our set into two equivalent parts to be polynomial-time or even logspace decidable. Glaßer et al. [GPSZ08] showed that for polynomial-time many-one reductions, autoreducibility and mitoticity are equivalent for non-trivial sets.

They follow the "trace" of an autoreduction for polynomially many steps in order to consistently decide whether the input belongs to the separator set or not. This approach is difficult to translate to the logspace setting, because here, the autoreduction can only be followed for a constant number of steps. We show that we can shift some of the complexity into the separator such that after constantly many steps of the autoreduction, we change the membership to the separator. In order to reduce the one part of the set to the other, we only need to consider the next few elements on the trace of the autoreduction, which shows that our set is weakly mitotic. In the logspace setting, we obtain new weak mitoticity results for P, Δ_k^{p}, and NEXP, and in the polynomial-time setting, we obtain new weak mitoticity results for PSPACE.

1.4 Publications and Bibliographical Remarks

This thesis is based on a technical report [GRW09] and the following publications:

[GW14]	C. Glaßer and M. Witek. Autoreducibility and mitoticity of logspace-complete sets for NP and other classes. In *Proceedings 39th International Symposium of Mathematical Foundations of Computer Science (MFCS)*, volume 8635 of *Lecture Notes in Computer Science*, pages 311–323. Springer, 2014.
[GNR+13a]	C. Glaßer, D. T. Nguyen, C. Reitwießner, A. L. Selman, and M. Witek. Autoreducibility of complete sets for log-space and polynomial-time reductions. In *Proceedings 40th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 7965 of *Lecture Notes in Computer Science*, pages 473–484. Springer, 2013.
[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 Symposium on Theoretical Informatics (LATIN)*, volume 7256 of *Lecture Notes in Computer Science*, pages 338–349. Springer, 2012.
[GRW11a]	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 fuer Informatik, 2011.
[GRSW10a]	C. Glaßer, C. Reitwießner, H. Schmitz, and M. Witek. Approximability and hardness in multi-objective optimization. In *Proceedings 6th Conference on Computability in Europe (CiE)*, volume 6158 of *Lecture Notes in Computer Science*, pages 180–189. Springer, 2010.

For each conference publication, a technical report can be found online [GW13, GNR+13b, FGL+11, GRW11b, GRSW10b]. The above papers relate to this thesis as follows.

Chapter 3 uses the framework we developed in [GRSW10a, FGL+12] to give a precise definition of multiobjective problems and their approximation. Chapters 4 and 5 summarize the results of our work on multiobjective traveling salesman problems [GRW09, GRW11a]. Chapter 6 is based on unpublished joint work with Christian Reitwießner on deterministic versions of the results of Chapter 5. Chapters 8 to 10 contain results from our paper about autoreducibility and mitoticity [GNR+13a], and Chapter 11 shows our most recent results on weak mitoticity [GW14].

Our work on autoreducibility and mitoticity was initiated by Dung T. Nguyen and Alan L. Selman [NS12], who showed autoreducibility results for NEXP and further noted that similar techniques lead to mitoticity results for PSPACE and EXP. A complete proof of these results was obtained in joint work [GNR$^+$13a]. In Sections 10.2 and 10.3 we give a shorter proof of these results.

Chapter 2

Preliminaries

2.1 Basic Mathematical Notations

Sets, Words and Numbers We denote by $\mathbb{N} = \{0, 1, 2, \dots\}$ and $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$ the set of natural numbers and the set of integers. For the set of positive natural numbers we use $\mathbb{N}^+ = \{1, 2, \dots\}$. We denote by $\mathbb{Q}$ and $\mathbb{R}$ the set of rational numbers and the set of real numbers. Moreover, we use standard notation to denote intervals of numbers, so for instance we write $[a, b) := \{x \mid a \leq x < b\}$. In some cases, we will use intervals of integers, and in some cases, we will use intervals of rational or real numbers. The difference will always be clear from the context. Let x be a number. We will use $\mathrm{abs}(x)$ to denote the absolute value, and analogously we will use $\mathrm{sgn}(x)$ to denote the sign of x. Moreover, we use $\lceil x \rceil := \min\{n \in \mathbb{Z} \mid n \geq x\}$ and $\lfloor x \rfloor := \max\{n \in \mathbb{Z} \mid n \leq x\}$.

For a finite alphabet Σ, let Σ^* denote the set of all words over Σ, including the empty word ε. For $w \in \Sigma^*$, let $|w|$ denote the length of w. If x is an integer, then let $|x|$ denote the length of the binary representation of x. For $n \in \mathbb{N}$ we will sometimes use the notation $\Sigma^{\geq n}$, $\Sigma^{\leq n}$, and Σ^n to denote the set of words over Σ of length at least n, at most n, and equal to n, respectively. We use $\overline{A}$ to denote the complement of the set A (with respect to some base set B, which will always be clear from the context). For a set $A \subseteq \Sigma^*$ we define the *characteristic function c_A of A* and the *semicharacteristic function χ_A of A* by

$$c_A \colon \Sigma^* \to \{0, 1\} \qquad c_A(x) = \begin{cases} 1 & \text{if } x \in A, \text{ and} \\ 0 & \text{if } x \notin A, \end{cases}$$

$$\chi_A \colon \Sigma^* \to \{0, 1\} \qquad \chi_A(x) = \begin{cases} 1 & \text{if } x \in A, \text{ and} \\ \text{not defined} & \text{if } x \notin A, \end{cases}$$

for all x. For $A \subseteq \mathbb{N}$ we define c_A and χ_A over $\mathbb{N}$ analogously. For sets A and B we denote the set difference of A and B by $A - B = \{x \in A \mid x \notin B\}$. Moreover, we use $A \triangle B = \{x \in A \cup B \mid c_A(x) \neq c_B(x)\}$ to denote the symmetric difference operator on sets. We denote the cardinality of the set A by $|A|$. Moreover, we call A *non-trivial* if neither A nor $\overline{A}$ are finite. In some cases we will use $2^A = \{B \mid B \subseteq A\}$ to denote the power set of A.

Encodings of Further Objects We will sometimes work with further objects such as graphs, sets or sequences. We will use an encoding function $\langle \cdot \rangle$ to encode such objects into integers. For a sequence $a_1, \dots, a_n$ of integers, the binary representation of $\langle a_1, \dots, a_n \rangle$ can be obtained by doubling the bits in the binary representations of each a_i and appending the results to a single bit string, where we use the bit string 10 as a separation block and as start and end marker. For further objects, we assume that an appropriate encoding into integers exists. We can hence use objects and lists of objects as input to our algorithms, where we define the length of an object or a list of objects as the length of its encoding.

Functions and Landau Notation For functions f and g, we define the composition $f \circ g$ by $(f \circ g)(x) = f(g(x))$. For a total function f, we will use $f^{(i)}$ to denote the i-th iteration of f, which is defined inductively by $f^{(0)}(x) = x$ for all x, and $f^{(i+1)} = (f \circ f^{(i)})$ for all $i \in \mathbb{N}$. Moreover, let log denote the logarithm to base 2. For simplicity, we define $\log(x) = 0$ for all $x < 1$, hence log and the iterated logarithm $\log^{(i)}$ for all i are total functions. For a total function f, we will use the *Landau notation* to denote the following classes of functions.

$$\begin{aligned}
&O(f) := \{g\colon \mathbb{N} \to \mathbb{N} \mid g \text{ is total and } \exists c \in \mathbb{N} \text{ such that } \forall n > c \text{ it holds that } g(n) \leq c \cdot f(n)\} \\
&\Omega(f) := \{g\colon \mathbb{N} \to \mathbb{N} \mid g \text{ is total and } \exists c \in \mathbb{N} \text{ such that } \forall n > c \text{ it holds that } g(n) \geq c \cdot f(n)\} \\
&\Theta(f) = O(f) \cap \Omega(f)
\end{aligned}$$

Moreover, we use $2^{O(f)} := \{g\colon \mathbb{N} \to \mathbb{N} \mid \exists h \in O(f)\colon \forall x \in \mathbb{N}\colon g(x) \leq 2^{h(x)}\}$.

2.2 Graph Theory

Graphs We define a *graph* as a tuple $G = (V, E)$, where V is a set of *vertices* and E is a set of *edges*. Throughout the first part of this thesis we will only consider graphs with finite V and E. We distinguish between a *directed graph*, where $E \subseteq \{(u, v) \mid u, v \in V\}$, an *undirected graph*, where $E \subseteq \{\{u, v\} \mid u, v \in V\}$, and a *multigraph*, where $E \subseteq \{(\{u, v\}, i) \mid u, v \in V, i \in \mathbb{N}\}$. In directed graphs, an edge (u, v) connects the vertices u and v and has the direction from u to v, while in an undirected graph and in a multigraph, the edge $\{u, v\}$ and the edge $(\{u, v\}, i)$ connect the vertices without a direction. The last component i of an edge of a multigraph is used to distinguish different edges that connect the same pair of vertices. An edge e that connects vertices u and v is said to be *incident to u (and v)*, and we define $[e] = \{u, v\}$. In this case we also say that *e covers u (and v)*. Moreover, if E is an edge set and u is a vertex, we say that E *covers* u if there exists an edge $e \in E$ such that e covers u. We say that a graph $G' = (V', E')$ is a *subgraph* of a graph $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. A directed graph is called *loop-free* if it does not contain an edge of the form (u, u), and an undirected graph where every edge connects two distinct vertices is called *simple*. A graph is called *complete* if it is a directed, loop-free graph, and for every pair of distinct vertices u, v there exists an edge from u to v, or if it is an undirected, simple graph, and every pair of distinct vertices is connected. For a graph $G = (V, E)$, $v \in V$, and $F \subseteq E$, we define the *degree of v in F* by $\deg_F(v) = |\{e \in F \mid e \text{ is incident to } v\}|$. Moreover, we denote the *degree of v in G* by $\deg_G(v) = \deg_E(v)$. A graph $G = (V, E)$ is called *bipartite* if there exists a partition of V into two disjoint sets V_1 and V_2 such that for every $e \in E$ it holds that $|[e] \cap V_1| = |[e] \cap V_2| = 1$.

Walks, Paths, Cycles Let $G = (V, E)$ be a graph. An sequence $v_0, e_1, v_1, e_2, \ldots, e_m, v_m$ that alternates between vertices $v_0, \ldots, v_m \in V$ and edges $e_1, \ldots, e_m \in E$ is called *walk (from v_0 to v_m)* if for all $1 \leq i \leq m$ it holds that $[e_i] = \{v_{i-1}, v_i\}$, and furthermore $e_i = (v_{i-1}, v_i)$ (if G is directed). The vertices v_0 and v_m are called the *endpoints* of the walk. A *closed walk* is a walk with $v_0 = v_m$, a *spanning walk* is a walk whose edges cover all vertices of G, a *path* is a walk without repeated vertices, and a *cycle* is a closed walk without repeated edges and vertices (except the endpoints) and at least two distinct vertices. Spanning cycles and spanning paths are also called *Hamiltonian.* If G is undirected or a multigraph, then we call G *connected* if for every pair of vertices u, v of G there exists a walk from u to v in G. If G is directed, then we call G *weakly connected* if G is connected if we consider it as an undirected graph, and we call G *strongly connected* if for every pair of vertices (u, v) there exists a walk from u to v in G. A connected graph G is called *Eulerian* if $\deg_G(v)$ is even for all vertices v of G. It is well-known that the edge set of an Eulerian graph corresponds to a closed spanning walk of the graph that can easily be constructed from the graph.

Trees Let $G = (V, E)$ be a graph, let $T \subseteq E$, and let $U \subseteq V$ be the set of vertices covered by T. Then T is called *tree* if (U, T) is (weakly) connected and does not contain a cycle or an edge e with $[e] = \{u\}$. If G is directed, then we further require that if $T \neq \emptyset$, then there exists exactly one vertex $u \in U$ called *root* such that for all $v \in U$ there exists a unique walk from u to v in (U, T). If $U = V$, then T is called *spanning tree (of G)*.

Matchings Let $G = (V, E)$ be a graph and let $M \subseteq E$. We call M a *matching (of G)* if for all $v \in V$ it holds that $\deg_M(v) \leq 1$. If M is a matching (of G), then we say that $v \in V$ is *matched by M* if $\deg_M(v) = 1$. If every vertex of G is matched by M, then M is called *perfect*. Clearly, if M is a perfect matching of G, then G has an even number of vertices, and $|M| = {}^{|V|}/_{2}$. A matching is called *near-perfect* if exactly one vertex is not matched. We will slightly change this notion in such a way that if M is a matching of G and $r \in \mathbb{N}$, then we say that M is *r-near perfect* if $|M| + r \geq {}^{|V|}/_{2}$. Hence, a matching is perfect if and only if it is 0-near perfect.

Cycle Covers For the graph $G = (V, E)$, a set $C = \{C_1, \dots, C_m\}$ with $m \in \mathbb{N}$ and $C_1, \dots, C_m \subseteq E$ is called *cycle cover of G* if each $C_i \in C$ is a cycle of G and for every $v \in V$ there exists exactly one $C_i \in C$ such that v is covered by C_i. Clearly, if C is a cycle cover of G, then C uses exactly $|V|$ edges. If C is a set of paths and cycles of G such that for every $v \in V$ there exists at most one path or cycle in C that covers v, then C is called *partial cycle cover*. For $r \in \mathbb{N}$ we define an *r-near cycle cover* to be a partial cycle cover that uses at least $|V| - r$ edges. Hence, C is a cycle cover of G if and only if C is a 0-near cycle cover. Observe that Hamiltonian cycles can be interpreted as special cycle covers consisting of a single cycle. For the sake of simplicity we will often consider cycle covers and partial cycle covers as sets of edges instead of sets of paths and cycles.

Edge-Labeled Graphs Let $k \in \mathbb{N}$. An *$\mathbb{N}^k$-labeled graph* is a triple $G = (V, E, w)$, where (V, E) is a graph, and $w\colon E \to \mathbb{N}^k$ is a total *labeling function*. We also refer to w as the *weight function (of G)*. For an $\mathbb{N}^k$-labeled graph $G = (V, E, w)$, we extend w to walks, edge sets, and sets of edge sets as follows. For a walk $W = v_0, e_1, v_1, \dots, e_m, v_m$ of G, we define $w(W) = \sum_{i \in \{1,\dots,m\}} w(e_i)$, for a set $E' \subseteq E$, we define $w(E') = \sum_{e \in E'} w(e)$, and for a set of edge sets $F = \{E_1, \dots, E_m\}$ with $E_i \subseteq E$ for all $1 \leq i \leq m$, we define $w(F) = \sum_{i \in \{1,\dots,m\}} w(E_i)$. Hence, if R is a walk, path, cycle, tree, matching, or (partial) cycle cover of G, then $w(R)$ denotes the weight of R.

2.3 Boolean Formulas

We will consider *propositional variables* over the values 1 (for *true*) and 0 (for *false*). Let V be a finite set of propositional variables. A *literal over V* is a propositional variable $v \in V$ or its *negation*, denoted by $\overline{v}$. A *clause over V* is a set of literals over V, and a *formula in conjunctive normal form (CNF, for short) over V* is a set of clauses over V. A *truth assignment over V* is a total function $I\colon V \to \{0, 1\}$.

Let I be a truth assignment over V, and let $v \in V$ be a variable. We say that I *satisfies* v if $I(v) = 1$, and I *satisfies* $\overline{v}$ if $I(v) = 0$. For a clause C over V, we say that I *satisfies* C if there exists a literal $l \in C$ such that I satisfies l, and for a formula F in CNF we say that I *satisfies* F if I satisfies all clauses $C \in F$. A formula F in CNF over V is called *satisfiable* if there exists a truth assignment over V that satisfies F.

We omit V if it is clear from the context.

2.4 Complexity Theory

2.4.1 Turing Machines and Transducers

Throughout this thesis we will use a standard Turing machine model that consists of a read-only input tape, $k \geq 1$ working tapes with infinite space to both sides, a finite set of states, including a starting state and accepting and possibly rejecting states, and a finite program that controls each step of the Turing machine. For a formal definition we refer to standard textbooks about computational complexity (see for instance the textbooks by Papadimitriou [Pap94] or by Arora and Barak [AB09]).

Terminology and Acceptance Behavior Let M be some Turing machine and x be some input. A *configuration* of M on input x is a complete description of some situation of M on input x, consisting of information about its state, its position on the input tape, and its position on the working tapes and their contents. The *start configuration* of M on input x is the configuration where the input tape contains x with the read-only head on the first symbol of x, where all working tapes are empty, and where M is in its starting state. An *accepting configuration* of M on input x is a configuration with an accepting state, and a *rejecting configuration* of M on input x is a configuration with a rejecting state. We will sometimes refer to the accepting and rejecting configurations as the *stop configurations.*

The configurations of M on some particular input x form a directed graph, where the nodes are the configurations of M on input x, and there exists an edge from configuration C_1 to configuration C_2 if and only if the Turing machine can reach the configuration C_2 from C_1 in one step, and C_1 is not a stop configuration. We will refer to this graph as the *configuration graph* of M on input x and denote it by $M(x)$. In some cases, we will also use $M(x)$ to denote the computation of M on input x.

A *computation path* (or simply a *computation*) of M on input x is a path in the configuration graph of M on input x that starts with the start configuration of M on input x. A computation or computation path is called *accepting* if it ends with an accepting configuration, and it is called *rejecting* if it ends with a rejecting configuration. We say that M *accepts* x if there exists an accepting computation path in the computation graph of M on input x, and we denote $L(M) = \{x \in \Sigma^* \mid M \text{ accepts } x\}$ as the *language accepted by* M.

We will also refer to Turing machines as described above as *nondeterministic*, because in every situation, more than one instruction might be applicable. Throughout this thesis we will assume that in every situation at most two instructions are applicable. If the program of the Turing machine has the property that in every situation at most one instruction of the program is applicable, then we call the Turing machine *deterministic.* If M is deterministic and for every input x there exists either an accepting or a rejecting computation path, then we say that M *decides* $L(M)$.

We further consider Turing machines that can output words and hence compute functions. A *Turing transducer* is a deterministic Turing machine M with no rejecting states that additionally has write-only access to an output tape. Initially, the output tape is empty, and in each step, the transducer can additionally append one symbol to its current output. We say that M *computes the function* f if for all input words x the following holds:

- If $f(x)$ is defined, then the computation of M on input x is accepting, and in the accepting configuration, the output tape of M contains $f(x)$.
- If $f(x)$ is not defined, then the computation of M on input x is not accepting.

Time and Space Bounds Let $s, t\colon \mathbb{N} \to \mathbb{N}$ be total functions and consider an arbitrary k-tape Turing machine or transducer M.

1. We say that M *works in space* s if for all but finitely many inputs x and for every computation path of M on input x it holds that in each configuration on the path, M uses at most $s(|x|)$ cells on its working tapes. Note that we count neither the space on the input nor the space on the output tape, if applicable.
2. We say that M *works in time* t if for all but finitely many inputs x and for every computation path of M on input x it holds that M terminates after at most $t(|x|)$ steps by reaching a stop configuration.

We will sometimes concentrate on the following space or time bounds.

1. The function s is called *space-constructible* if there exists a deterministic Turing machine that for every $n \in \mathbb{N}$ stops on input 1^n and uses exactly space $s(n)$.
2. The function t is called *time-constructible* if there exists a deterministic Turing machine that for every $n \in \mathbb{N}$ stops on input 1^n after exactly $t(n)$ steps.

2.4.2 Complexity Classes and Function Classes

We define the following generic complexity classes and function classes. Let $s, t\colon \mathbb{N} \to \mathbb{N}$ be total functions.

$$\begin{aligned}
\mathrm{DSPACE}(s) &:= \{L(M) \mid M \text{ is a deterministic Turing machine}\\ &\qquad\qquad\quad \text{that works in space } s\}\\
\mathrm{NSPACE}(s) &:= \{L(M) \mid M \text{ is a nondeterministic Turing machine}\\ &\qquad\qquad\quad \text{that works in space } s\}\\
\mathrm{DTIME}(t) &:= \{L(M) \mid M \text{ is a deterministic Turing machine}\\ &\qquad\qquad\quad \text{that works in time } t\}\\
\mathrm{NTIME}(t) &:= \{L(M) \mid M \text{ is a nondeterministic Turing machine}\\ &\qquad\qquad\quad \text{that works in time } t\}\\
\mathrm{FSPACE}(s) &:= \{f \mid \text{there is a Turing transducer that computes } f\\ &\qquad\quad \text{and works in space } s\}\\
\mathrm{FTIME}(t) &:= \{f \mid \text{there is a Turing transducer that computes } f\\ &\qquad\quad \text{and works in time } t\}
\end{aligned}$$

Moreover, for a class of functions $\mathcal{F}$ we define

$$\begin{aligned}
\mathrm{DSPACE}(\mathcal{F}) &:= \bigcup_{f\in\mathcal{F}} \mathrm{DSPACE}(f) & \mathrm{DTIME}(\mathcal{F}) &:= \bigcup_{f\in\mathcal{F}} \mathrm{DTIME}(f)\\
\mathrm{NSPACE}(\mathcal{F}) &:= \bigcup_{f\in\mathcal{F}} \mathrm{NSPACE}(f) & \mathrm{NTIME}(\mathcal{F}) &:= \bigcup_{f\in\mathcal{F}} \mathrm{NTIME}(f)\\
\mathrm{FSPACE}(\mathcal{F}) &:= \bigcup_{f\in\mathcal{F}} \mathrm{FSPACE}(f) & \mathrm{FTIME}(\mathcal{F}) &:= \bigcup_{f\in\mathcal{F}} \mathrm{FTIME}(f)
\end{aligned}$$

which enables us to use the Landau notation in the definition of complexity classes as well.

Prominent Space Classes We will mainly work on the following space classes.

$$\begin{aligned}
\mathrm{FL} &:= \mathrm{FSPACE}(O(\log n)) & & \\
\mathrm{L} &:= \mathrm{DSPACE}(O(\log n)) & \mathrm{NL} &:= \mathrm{NSPACE}(O(\log n)) \\
\mathrm{LIN} &:= \mathrm{DSPACE}(O(n)) & \mathrm{NLIN} &:= \mathrm{NSPACE}(O(n)) \\
\mathrm{PSPACE} &:= \bigcup_{k\in\mathbb{N}} \mathrm{DSPACE}(O(n^k)) & \mathrm{NPSPACE} &:= \bigcup_{k\in\mathbb{N}} \mathrm{NSPACE}(O(n^k))
\end{aligned}$$

Note that by Savitch's Theorem [Sav70], $\mathrm{NSPACE}(s) \subseteq \mathrm{DSPACE}(s^2)$ for every space-constructible function $s \geq \log$, hence PSPACE = NPSPACE. Moreover, by linear compression we can scale down the space required on working tapes of Turing machines by arbitrary constant factors, which means that $\mathrm{FSPACE}(O(s)) = \mathrm{FSPACE}(s)$, $\mathrm{DSPACE}(O(s)) = \mathrm{DSPACE}(s)$ and $\mathrm{NSPACE}(O(s)) = \mathrm{NSPACE}(s)$. So we obtain:

$$\begin{aligned}
\mathrm{FL} &= \mathrm{FSPACE}(\log n) & \mathrm{L} &= \mathrm{DSPACE}(\log n) & \mathrm{NL} &= \mathrm{NSPACE}(\log n) \\
\mathrm{LIN} &= \mathrm{DSPACE}(n) & \mathrm{NLIN} &= \mathrm{NSPACE}(n) & \mathrm{PSPACE} &= \bigcup_{k\in\mathbb{N}} \mathrm{DSPACE}(n^k)
\end{aligned}$$

The functions in FL are called *logarithmic-space computable*. We will use *logspace* as a synonym for logarithmic-space.

Prominent Time Classes Analogously we will consider the following time classes.

$$\begin{aligned}
\mathrm{FP} &:= \bigcup_{k\in\mathbb{N}} \mathrm{FTIME}(O(n^k)) & & \\
\mathrm{P} &:= \bigcup_{k\in\mathbb{N}} \mathrm{DTIME}(O(n^k)) & \mathrm{NP} &:= \bigcup_{k\in\mathbb{N}} \mathrm{NTIME}(O(n^k)) \\
\mathrm{E} &:= \mathrm{DTIME}(2^{O(n)}) & \mathrm{NE} &:= \mathrm{NTIME}(2^{O(n)}) \\
\mathrm{EXP} &:= \bigcup_{k\in\mathbb{N}} \mathrm{DTIME}(2^{O(n^k)}) & \mathrm{NEXP} &:= \bigcup_{k\in\mathbb{N}} \mathrm{NTIME}(2^{O(n^k)})
\end{aligned}$$

The functions in FP are called *polynomial-time computable*.

Complements of Complexity Classes For a complexity class $\mathcal{C}$ we define

$$\mathrm{co}\mathcal{C} := \{L \mid \overline{L} \in \mathcal{C}\}$$

as the class of languages whose complements are contained in $\mathcal{C}$. Immerman [Imm88] and Szelepcsényi [Sze88] showed that for $s \geq \log$ it holds that $\mathrm{NSPACE}(s) = \mathrm{coNSPACE}(s)$, so in particular we have NL = coNL and NLIN = coNLIN.

2.4.3 Alternating Turing Machines

An *alternating Turing machine* is a generalization of a Turing machine, where we additionally partition the states into *universal* and *existential* states. Let M be an alternating Turing machine and let x be some input. We consider the configuration graph of M on input x and call a configuration *existential* if its state is existential and *universal* if its state is universal. We define *accepting subgraphs* of the configuration graph inductively as follows.

1. If C is an accepting configuration, then $(\{C\}, \emptyset)$ is an accepting subgraph with root C.

2. If C is an existential configuration that has a successor configuration C' in the configuration graph, and C' is the root of an accepting subgraph (V, E), then $(V \cup \{C\}, E \cup \{(C, C')\})$ is an accepting subgraph with root C.
3. If C is a universal configuration that has successor configurations C_1 and C_2 in the configuration graph such that C_1 is the root of an accepting subgraph (V_1, E_1) and C_2 is the root of an accepting subgraph (V_2, E_2), then $(V_1 \cup V_2 \cup \{C\}, E_1 \cup E_2 \cup \{(C, C_1), (C, C_2)\})$ is an accepting subgraph with root C.

We say that *an alternating Turing machine accepts x* if its start configuration is the root of an accepting subgraph, and use $L(M)$ to denote the set of words accepted by an alternating Turing machine M. Note that we can think of nondeterministic Turing machines as alternating Turing machines with only existential states. We use space and time bounds for alternating Turing machines analogously to those for nondeterministic Turing machines. We define the following generic complexity classes, where $s, t \colon \mathbb{N} \to \mathbb{N}$ are total functions.

$$\begin{aligned} \mathrm{ASPACE}(s) &:= \{L(M) \mid M \text{ is an alternating Turing machine that works in space } s\} \\ \mathrm{ATIME}(t) &:= \{L(M) \mid M \text{ is an alternating Turing machine that works in time } t\} \end{aligned}$$

Again we extend our definition to classes of functions $\mathcal{F}$.

$$\mathrm{ASPACE}(\mathcal{F}) := \bigcup_{f \in \mathcal{F}} \mathrm{ASPACE}(f) \qquad \mathrm{ATIME}(\mathcal{F}) := \bigcup_{f \in \mathcal{F}} \mathrm{ATIME}(f)$$

We will use the complexity class AL $:=$ ASPACE$(O(\log n))$ $=$ ASPACE$(\log n)$. Chandra et al. [CKS81] showed that AL $=$ P, hence alternating logspace characterizes P. This relation between alternating logspace and polynomial time will be helpful in the following chapters.

2.4.4 Oracle Turing Machines

Oracle Machines and Oracle Access An *oracle Turing machine* is a Turing machine that further has a fixed number of oracle tapes and some additional states. Moreover, we assign an oracle set O to the oracle Turing machine. We generally require write-only access to the oracle tapes. In each step, an oracle Turing machine can additionally append a single symbol to the contents of some oracle tape or enter a special query state q_i (where i is the number of some oracle tape). If the Turing machine enters the query state q_i, then in the next step it will enter the special state q_y if the content of oracle tape i is contained in O, and it will enter the special state q_n if the content of the oracle tape i is not contained in O. In either case it will erase the oracle tape i. We will denote an oracle Turing machine M that uses the oracle set O by M^O and the language accepted by M with oracle O by $L(M^O)$. Moreover, if M is nondeterministic, then we require M to work deterministically whenever its oracle tapes are not empty.

Relativized Complexity Classes Note that for oracle Turing machines we can use the same definition of space and time bounds that we used without oracle access. We will not count the space used on oracle tapes, and we will count every oracle query as a single step. Hence we can define *relativized complexity classes* analogously to the complexity classes considered so far. Let O be an arbitrary set, let $t \colon \mathbb{N} \to \mathbb{N}$ be a total function, and let $\mathcal{C}$ be a complexity class.

$$\begin{aligned} \mathrm{DTIME}(t)^O &:= \{L(M^O) \mid M \text{ is a deterministic oracle Turing machine} \\ &\qquad\qquad \text{that works in time } t\} \\ \mathrm{DTIME}(t)^{\mathcal{C}} &:= \bigcup_{O \in \mathcal{C}} \mathrm{DTIME}(t)^O \end{aligned}$$

We have analogous definitions for the remaining space and time classes and obtain classes such as P^O or NP^O, for instance. We note at this point that P^O and NP^O are very robust against modifications of the oracle access mechanism. In particular, it makes no difference if we allow multiple oracle tapes or restrict ourselves to a single oracle tape. For relativized space classes, the situation is more complicated, and there is no canonical oracle access method. For instance, Lynch [Lyn78] showed that logspace oracle Turing machines become more powerful if they are allowed to use more oracle tapes. We use the general model proposed by Lynch [Lyn78] and restrict the number of oracle tapes whenever necessary.

The Polynomial-Time Hierarchy The *polynomial-time hierarchy* is a hierarchy of complexity classes between P and PSPACE, introduced by Stockmeyer [Sto76]. It includes NP and coNP and is inductively defined as follows, where $i \in \mathbb{N}$.

$$\begin{aligned}
\Sigma_0^{\mathrm{p}} &:= \Pi_0^{\mathrm{p}} := \Delta_0^{\mathrm{p}} := \mathrm{P}\\
\Delta_{i+1}^{\mathrm{p}} &:= \mathrm{P}^{\Sigma_i^{\mathrm{p}}}\\
\Sigma_{i+1}^{\mathrm{p}} &:= \mathrm{NP}^{\Sigma_i^{\mathrm{p}}}\\
\Pi_{i+1}^{\mathrm{p}} &:= \mathrm{co}\Sigma_{i+1}^{\mathrm{p}}
\end{aligned}$$

Moreover, we define $\mathrm{PH} := \bigcup_{i\in\mathbb{N}} \Sigma_i^{\mathrm{p}}$.

2.4.5 Reducibilities and Complete Problems

Reducibilities Let A and B be two sets. We will work with *reducibility notions* that are defined as follows.

1. *A is polynomial-time Turing reducible to B ($A \leq_{\mathrm{T}}^{\mathrm{p}} B$)* if and only if there exists a polynomial-time oracle Turing machine that accepts A with B as its oracle.
2. *A is polynomial-time* log-*Turing reducible to B ($A \leq_{\text{log-T}}^{\mathrm{p}} B$)* if and only if $A \leq_{\mathrm{T}}^{\mathrm{p}} B$ via an oracle Turing machine that asks at most $O(\log n)$ many queries.
3. *A is polynomial-time truth-table reducible to B ($A \leq_{\mathrm{tt}}^{\mathrm{p}} B$)* if and only if $A \leq_{\mathrm{T}}^{\mathrm{p}} B$ via an oracle Turing machine whose queries are nonadaptive (i.e., independent of the oracle).
4. *A is polynomial-time k-truth-table reducible to B ($A \leq_{k\text{-tt}}^{\mathrm{p}} B$)* if and only if $A \leq_{\mathrm{tt}}^{\mathrm{p}} B$ via an oracle Turing machine that asks at most k queries.
5. *A is polynomial-time bounded-truth-table reducible to B ($A \leq_{\mathrm{btt}}^{\mathrm{p}} B$)* if and only if $A \leq_{k\text{-tt}}^{\mathrm{p}} B$ for some k.
6. *A is polynomial-time α-truth-table reducible to B ($A \leq_{\alpha\mathrm{tt}}^{\mathrm{p}} B$)* if and only if there exists a polynomial-time computable function f such that for all x it holds that $f(x) = \langle y_1, \dots, y_k\rangle$ with $c_A(x) = \alpha(c_B(y_1), \dots, c_B(y_k))$, where α is a fixed k-ary Boolean function.
7. *A is polynomial-time disjunctive-truth-table reducible to B ($A \leq_{\mathrm{dtt}}^{\mathrm{p}} B$)* if and only if there exists a polynomial-time computable function f such that for all x, $f(x) = \langle y_1, \dots, y_n\rangle$ for some $n \geq 1$ and $c_A(x) = \max\{c_B(y_1), \dots, c_B(y_n)\}$.
8. *A is polynomial-time k-disjunctive-truth-table reducible to B ($A \leq_{k\text{-dtt}}^{\mathrm{p}} B$)* if and only if $A \leq_{\mathrm{dtt}}^{\mathrm{p}} B$ via some f such that $f(x) = \langle y_1, \dots, y_n\rangle$ implies $n \leq k$.
9. *A is polynomial-time bounded-disjunctive-truth-table reducible to B ($A \leq_{\mathrm{bdtt}}^{\mathrm{p}} B$)* if and only if $A \leq_{k\text{-dtt}}^{\mathrm{p}} B$ for some k.
10. *A is polynomial-time conjunctive-truth-table reducible to B ($A \leq_{\mathrm{ctt}}^{\mathrm{p}} B$)* if and only if there exists a polynomial-time computable function f such that for all x, $f(x) = \langle y_1, \dots, y_n\rangle$ for some $n \geq 1$ and $c_A(x) = \min\{c_B(y_1), \dots, c_B(y_n)\}$.

11. *A is polynomial-time k-conjunctive-truth-table reducible to B ($A \leq^{\mathrm{p}}_{k\text{-ctt}} B$)* if and only if $A \leq^{\mathrm{p}}_{\mathrm{ctt}} B$ via some f such that $f(x) = \langle y_1, \dots, y_n \rangle$ implies $n \leq k$.
12. *A is polynomial-time bounded-conjunctive-truth-table reducible to B ($A \leq^{\mathrm{p}}_{\mathrm{bctt}} B$)* if and only if $A \leq^{\mathrm{p}}_{k\text{-ctt}} B$ for some k.
13. *A is polynomial-time many-one reducible to B ($A \leq^{\mathrm{p}}_{\mathrm{m}} B$)* if and only if there exists some $f \in \mathrm{FP}$ such that $c_A(x) = c_B(f(x))$ for all x.

We also use the following *logspace reducibility notions*, which are defined analogously in terms of logspace oracle Turing machines and logspace-computable functions:

$$\leq^{\log}_{\mathrm{T}}, \leq^{\log}_{\log\text{-T}}, \leq^{\log}_{\mathrm{tt}}, \leq^{\log}_{k\text{-tt}}, \leq^{\log}_{\mathrm{btt}}, \leq^{\log}_{\alpha\mathrm{tt}}, \leq^{\log}_{\mathrm{dtt}}, \leq^{\log}_{k\text{-dtt}}, \leq^{\log}_{\mathrm{bdtt}}, \leq^{\log}_{\mathrm{ctt}}, \leq^{\log}_{k\text{-ctt}}, \leq^{\log}_{\mathrm{bctt}}, \leq^{\log}_{\mathrm{m}}$$

Observe that for polynomial-time bounded reducibility notions and for truth-table reducibility notions, the number of oracle tapes we use is not relevant, so here we will always assume that we use only one oracle tape. In the general case of logspace-bounded Turing reductions, we will further consider the number of oracle tapes that we use. So we have the following additional definitions.

1. *A is logspace k-tape Turing reducible to B ($A \leq^{\log[k]}_{\mathrm{T}} B$)* if and only if $A \leq^{\log}_{\mathrm{T}} B$ via some oracle Turing machine with k oracle tapes.
2. *A is logspace k-tape* log-*Turing reducible to B ($A \leq^{\log[k]}_{\log\text{-T}} B$)* if and only if $A \leq^{\log}_{\log\text{-T}} B$ via some oracle Turing machine with k oracle tapes.

Ladner and Lynch [LL76] showed that $A \leq^{\log[1]}_{\mathrm{T}} B$ if and only if $A \leq^{\log}_{\mathrm{tt}} B$.

Moreover, we will sometimes use *poly-logspace reducibilities*, which are defined analogously to logspace reducibilities, except that we have a space bound of $\log^k$ for some $k \in \mathbb{N}$.

Equivalence Let $\leq$ be an arbitrary reducibility notion and let A and B be sets. We say that *A is $\leq$-equivalent to B ($A \equiv B$)* if and only if $A \leq B$ and $B \leq A$.

Hardness and Completeness Let $\mathcal{C}$ be some complexity class and let $\leq$ be some reducibility notion. Let A be a set. We say that A is *$\leq$-hard for $\mathcal{C}$* if and only if for every $B \in \mathcal{C}$ it holds that $B \leq A$. If additionally it holds that $A \in \mathcal{C}$, then we say that *A is $\leq$-complete for $\mathcal{C}$*.

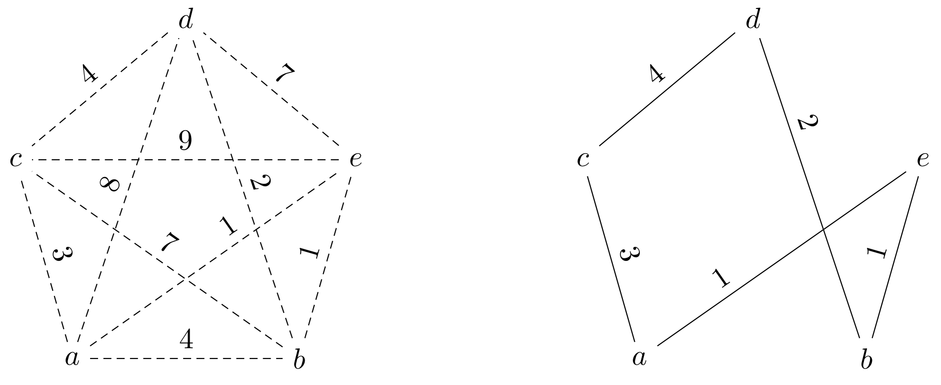
Part I

Multiobjective Traveling Salesman Problems

Chapter 3

Multiobjective Optimization

In the first part of this thesis we study the approximability of the traveling salesman problem and its most important multiobjective variants. This problem is defined on complete graphs with edge weights, and we want to find a Hamiltonian cycle (i.e., a cycle that visits every vertex exactly once) of minimal or maximal weight. Figure 3.1 shows an example of an instance of the traveling salesman problem and a minimal Hamiltonian cycle.



a) Complete graph K_5 b) Hamiltonian cycle in K_5

Figure 3.1: Part a) shows the complete graph K_5 with edge weights in $\mathbb{N}$. Part b) shows a minimum weight Hamiltonian cycle with overall weight 11.

In most real-world scenarios, however, we have to deal with multiple objectives at the same time. For instance, in a typical application of the traveling salesman problem, we want to find the shortest roundtrip for a given set of cities, but at the same time, we want to minimize further objectives, such as travel costs or also travel time. We can easily modify our problem definition by considering complete graphs with edge weights in $\mathbb{N}^k$, where k is the number of objectives, but solving the problem becomes more complicated. Already in the case $k = 2$, different Hamiltonian cycles can be incomparable, and a *unique* optimal solution does not exist.

Pareto Sets and Approximation We use the concept of Pareto sets to cope with this difficulty. Suppose we have a complete graph with edge weights in $\mathbb{N}^k$, and we want to find a Hamiltonian cycle that has a low weight in all objectives. We call a Hamiltonian cycle Pareto-optimal if no other Hamiltonian cycle exists that is at least as good in all objectives and even better in at least one objective. Such cycles are optimal tradeoffs between the different objectives, because we cannot improve them in one objective without degrading them in another objective again. The Pareto set is defined as the set of all Pareto-optimal Hamiltonian cycles, and the general idea in multiobjective optimization is to compute the Pareto set and present it to the user, who can then choose some optimal tradeoff. Figure 3.2 demonstrates an example of such a Pareto set.

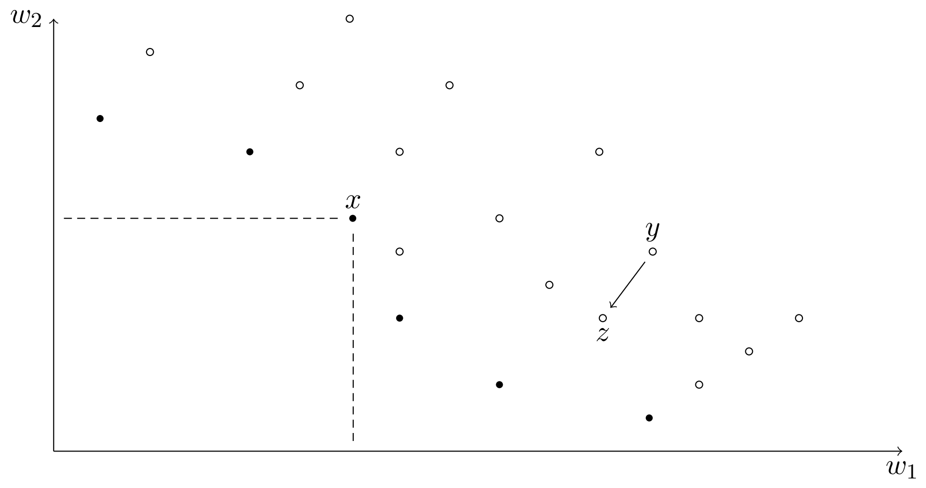


Figure 3.2: Example of the solution value space of a two-objective minimization problem. Every point corresponds to the value of one or more solutions (in the case of the traveling salesman problem, Hamiltonian cycles) with components w_1 and w_2. The solution z is better than the solution y in both objectives, hence y is not Pareto-optimal. The solutions x and z are incomparable, because $w_1(x) < w_1(z)$ and $w_2(z) < w_2(x)$. However, x is Pareto-optimal, because there does not exist a solution to the left and below x, so the values of x cannot be improved any further. The solid points are the values of the solutions in the Pareto set.

Recall that already in the case of a single-objective, the traveling salesman problem is NP-hard. This in particular means that we cannot expect to compute the Pareto set in the case of multiple objectives, so again we work on approximations. It is quite straightforward to generalize the concept of approximations to the multiobjective case. We call a set of solutions an α-approximate Pareto set, if for every Pareto-optimal solution there exists a solution in our set that α-approximates the Pareto-optimal solution. In general, we will compute such approximate Pareto sets. We will give a precise definition of this concept in Section 3.1.

Multiobjective Variants of the Traveling Salesman Problem In the single-objective case, the minimum traveling salesman problem does not admit an α-approximation for constant α, unless P = NP [SG76], so we concentrate on its metric variant. Recall that Christofides [Chr76] showed a $3/2$-approximation algorithm for this problem. We will analyze its approximability in the case of two objectives, and we will further study similar Hamiltonian path problems.

Moreover, we will study the approximability of maximization variants of the traveling salesman problem, where we want to find Hamiltonian cycles of maximum weight. We distinguish between directed and undirected graphs. In the single-objective case, Fisher, Nemhauser and Wolsey [FNW79] showed a basic approximation algorithm for the maximum traveling salesman problem that is based on cycle covers. They obtained a $1/2$ and a $2/3$ approximation for the case of directed and undirected graphs, respectively. We will demonstrate a non-trivial adaption of their approach to the multiobjective case and obtain improved approximation ratios.

We refer to Section 3.2 for a definition of the multiobjective problems we study, and we refer to Section 3.3 for a summary of the contributions we obtain.

3.1 Basic Definitions

We proceed with a general definition of multiobjective optimization concepts.

Multiobjective Problems We adapt the notation from Glaßer et al. [GRSW10a] and Fleszar et al. [FGL+12]. Let $k \geq 1$. A *k-objective* NP *optimization problem* (*k-objective problem*, for short) is a tuple $(S, f, \leftarrow)$ with the following properties.

- $S \colon \mathbb{N} \to 2^{\mathbb{N}}$ 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 $|s| \leq p(|x|)$, and we require $\{\langle x, s\rangle \mid x \in \mathbb{N}, s \in S^x\} \in \mathrm{P}$.
- $f \colon \{\langle x, s\rangle \mid x \in \mathbb{N}, s \in S^x\} \to \mathbb{N}^k$ 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 $(a_1, \dots, a_k) \leftarrow (b_1, \dots, b_k) \iff a_1 \leftarrow_1 b_1 \wedge \cdots \wedge a_k \leftarrow_k b_k$, 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 $f^x(s) = (v_1, \dots, v_k)$. 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, i.e. 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 $\mathrm{opt}_{\leftarrow} \colon 2^{\mathbb{N}^k} \to 2^{\mathbb{N}^k}$, $\mathrm{opt}_{\leftarrow}(M) = \{y \in M \mid \forall z \in M[z \leftarrow y \implies z = y]\}$ 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 (of x)* and contains all non-dominated solutions, and for a given problem instance x, our goal is to compute the Pareto set of x.

For better readability, when we define a k-objective problem, we will often omit an explicit definition of the corresponding tuple and explain its components instead. For each problem we consider there exists a simple encoding into natural numbers, so we can also define our problems on objects such as formulas or graphs instead of natural numbers. Below we give an example of such a problem definition.

Example 3.1.1 *The satisfiability problem on formulas in conjunctive normal form is defined as* CNF-SAT $= \{F \mid F$ *is a satisfiable Boolean formula in CNF*$\}$. *We can define a corresponding* 1*-objective* NP *optimization problem as follows:*

> ***Maximum Weighted Satisfiability***
> *Notation:* MaxSAT
> *Instance: formula F in CNF over a set of variables V and weight function $w \colon F \to \mathbb{N}$*
> *Solution: truth assignment $I \colon V \to \{0, 1\}$*
> *Objective: maximize $w(I) = \sum_{C \in F\colon I(C)=1} w(C)$*

This corresponds to the tuple definition MaxSAT $= (S, f, \geq)$*, where a problem instance x encodes a set of propositional variables V, a Boolean formula F in CNF over V, and a weight function $w \colon F \to \mathbb{N}$. Moreover, the function S maps the instance x to the set of all truth assignments I over V, and the function f maps the instance x and the truth assignment I to the sum of the weights of all clauses of F that are satisfied by I.*

Multiobjective Approximation We generalize the approximation concept to the multiobjective case as follows. Let $a \geq 1$. We define:

$$u \overset{a}{\leq} v \iff u \leq a \cdot v \qquad\qquad u \overset{a}{\geq} v \iff a \cdot u \geq v$$

Fix some $\leftarrow\, = (\leftarrow_1, \ldots, \leftarrow_k)$ where $\leftarrow_i \in \{\leq, \geq\}$, and let $p, q \in \mathbb{N}^k$ and $\alpha \in \mathbb{R}^k$ such that $\alpha_1, \ldots, \alpha_k \geq 1$. We further define

$$p \overset{\alpha}{\leftarrow} q \iff \bigwedge_{i=1}^{k} p_i \overset{\alpha_i}{\leftarrow}_i q_i,$$

hence $\overset{\alpha}{\leftarrow}$ always points to the better value, relaxed by a factor α.

Let $\mathcal{O} = (S, f, \leftarrow)$ be a k-objective problem, $x \in \mathbb{N}$ be a problem instance of $\mathcal{O}$, and $\alpha \in \mathbb{R}^k$ such that $\alpha_1, \ldots, \alpha_k \geq 1$. For solutions $s, t \in S^x$ we say that *s α-approximates t* if $f(s) \overset{\alpha}{\leftarrow} f(t)$. A set of solutions $S' \subseteq S^x$ is called *α-approximate solution set for x* if for every $t \in S^x$ there exists some $s \in S'$ such that s α-approximates t. If all α_i are equal, we may use the scalar α_1 instead of the vector α and call S' an *α_1-approximate solution set for x*. An α-approximate solution set for x is also called an *α-approximate Pareto set (of x)*.

Deterministic Approximation While Pareto sets are typically hard to compute (and in many cases can have exponential cardinality in the length of the instance x), it is often much easier to compute approximate Pareto sets. We will consider polynomial-time algorithms that compute good approximations. An algorithm is called *α-approximation algorithm for $\mathcal{O}$* if there exists a polynomial p such that for every input $x \in \mathbb{N}$, the algorithm computes an α-approximate solution set for x in time at most $p(|x|)$. An algorithm is called *polynomial-time approximation scheme (PTAS, for short) for $\mathcal{O}$* if there exists a polynomial p such that for every input $x \in \mathbb{N}$ and for every $\varepsilon > 0$, the algorithm computes a $(1+\varepsilon)$-approximate solution set for x in time at most $p(|x|)$. A PTAS is called *fully polynomial-time approximation scheme (FPTAS, for short) for $\mathcal{O}$* if its runtime is bounded by $p(|x| + 1/\varepsilon)$.

Randomized Approximation An algorithm is called *randomized* if it has bitwise read-only access to an arbitrary sequence of random bits. Hence, on some particular input x, if we execute a randomized algorithm twice, it may behave different during the second execution. In particular, it is possible that a randomized algorithm does not terminate at all.

An algorithm is called a *randomized α-approximation algorithm for $\mathcal{O}$* if there exists a polynomial p such that for every problem instance x of $\mathcal{O}$, the algorithm terminates after at most $p(|x|)$ steps if executed on x, and moreover, with probability at least $1/2$, it returns an α-approximate solution set for x. In case that the algorithm returns such an approximate solution set, we say that the algorithm *succeeds*, otherwise we say that the algorithm *fails*. Analogously we define the notions of *polynomial-time randomized approximation scheme (PRAS, for short)* and *fully polynomial-time randomized approximation scheme (FPRAS, for short)*.

We will construct new randomized algorithms that call existing randomized algorithms several times. In these cases we will often use *amplified* versions of the existing randomized algorithms, where the original algorithm is called several times in order to increase the overall success probability. Here, the following lemma will be useful.

Lemma 3.1.2 *For all $m \in \mathbb{N}^+$ it holds that $(1 - \frac{1}{2^m})^m \geq \frac{1}{2}$.*

Proof By induction we can show that $\frac{m}{2^m} \leq \frac{1}{2}$ for all $m \in \mathbb{N}^+$. This clearly holds for $m = 1$, and for $m \geq 1$ we further have

$$\frac{m+1}{2^{m+1}} = \frac{1}{2} \cdot \left(\frac{m}{2^m} + \frac{1}{2^m} \right) \leq \frac{1}{2} \cdot \left(\frac{m}{2^m} + \frac{m}{2^m} \right) = \frac{m}{2^m} \leq \frac{1}{2}$$

where the last step holds by the induction hypothesis. Bernoulli's inequality states that

$$(1+x)^m \geq 1 + mx$$

for all $x \in \mathbb{R}$ with $x \geq -1$ and $m \in \mathbb{N}$. Since $\frac{1}{2^m} \leq 1$ for all $m \in \mathbb{N}^+$ we obtain

$$\left(1 - \frac{1}{2^m}\right)^m \geq 1 - \frac{m}{2^m} \geq 1 - \frac{1}{2} = \frac{1}{2}.$$

□

Approximability of Problems We say that the k-objective problem $\mathcal{O}$ is *α-approximable* if there exists an α-approximation algorithm for $\mathcal{O}$. Analogously, we say that $\mathcal{O}$ is *randomized α-approximable* if there exists a randomized α-approximation algorithm for $\mathcal{O}$. When we work on maximization problems, i.e., $\mathcal{O} = (S, f, \geq)$, we will typically use an approximation factor β such that $0 < \beta < 1$. In this case, we say that $\mathcal{O}$ is β-approximable if it is α-approximable with $\alpha = 1/\beta$ in the way defined above.

Example 3.1.3 *Recall the definition of* MAXSAT *from Example 3.1.1 and consider an algorithm that on input $\langle V, F, w\rangle$, where V is a set of propositional variables, F is a formula in CNF over V, and $w\colon F \to \mathbb{N}$, returns the set $\{I_0, I_1\}$, where $I_0, I_1\colon V \to \{0,1\}$ are truth assignments with $I_0(v) = 0$ and $I_1(v) = 1$ for all $v \in V$. The algorithm runs in polynomial time in the length of the input. Moreover, let $I\colon V \to \{0,1\}$ be an arbitrary truth assignment over V. For every non-empty clause $C \in F$ it holds that at least one of I_0 and I_1 satisfies C, hence $w(I_0) + w(I_1) \geq w(I)$, and so $w(I_j) \geq 1/2 \cdot w(I)$ for some $j \in \{0,1\}$. Hence* MAXSAT *is $1/2$-approximable.*

3.2 Relevant Multiobjective Problems

Let $k \geq 1$. In the following we define the most relevant multiobjective problems whose approximation ratio we improve in this thesis. In Chapter 4 we will first consider a generalization of the classical minimum metric traveling salesman problem to the multiobjective case.

***k*-Objective Minimum Traveling Salesman on Multigraphs**

Notation:	MULTIGRAPH k-MINTSP
Instance:	$\mathbb{N}^k$-labeled undirected multigraph $G = (V, E, w)$
Solution:	closed spanning walk W
Objective:	minimize $w(W)$

Unlike the case studied in the literature so far (see for instance Manthey and Ram [MR09]), we consider multigraphs and do not restrict ourselves to weight functions that are metric in every component. We will later see that this case is more general and practically relevant. Analogously we will consider generalizations of the single-objective minimum traveling salesman path problems.

***k*-Objective Minimum Traveling Salesman Path on Multigraphs**

Notation:	MULTIGRAPH k-MINTSPP
Instance:	$\mathbb{N}^k$-labeled undirected multigraph $G = (V, E, w)$
Solution:	spanning walk W
Objective:	minimize $w(W)$

***k*-Objective Minimum Traveling Salesman s-Path on Multigraphs**

Notation:	MULTIGRAPH k-MINTSPPS
Instance:	$\mathbb{N}^k$-labeled undirected multigraph $G = (V, E, w)$ and $s \in V$
Solution:	spanning walk W, starting at s
Objective:	minimize $w(W)$

***k*-Objective Minimum Traveling Salesman s-t-Path on Multigraphs**

Notation:	MULTIGRAPH k-MINTSPPST
Instance:	$\mathbb{N}^k$-labeled undirected multigraph $G = (V, E, w)$ and $s, t \in V$
Solution:	spanning walk W, starting at s and ending at t
Objective:	minimize $w(W)$

In Chapter 5 and Chapter 6 we will shift our focus to generalizations of the maximization variant of the traveling salesman problem. We will distinguish between directed and undirected graphs and show results for the following problems.

***k*-Objective Maximum Asymmetric Traveling Salesman Problem**

Notation:	k-MAXATSP
Instance:	$\mathbb{N}^k$-labeled directed complete graph $G = (V, E, w)$
Solution:	Hamiltonian cycle $C \subseteq E$ of G
Objective:	maximize $w(C)$

***k*-Objective Maximum Symmetric Traveling Salesman Problem**

Notation:	k-MAXSTSP
Instance:	$\mathbb{N}^k$-labeled undirected complete graph $G = (V, E, w)$
Solution:	Hamiltonian cycle $C \subseteq E$ of G
Objective:	maximize $w(C)$

We will further show that the method we apply to k-MAXATSP and k-MAXSTSP is not restricted to these particular problems. For this purpose we show its application to a multiobjective satisfiability problem, which is defined as follows.

***k*-Objective Maximum Weighted Satisfiability**

Notation:	k-MAXSAT
Instance:	formula H in CNF over a set of variables V and weight function $w\colon H \to \mathbb{N}^k$
Solution:	truth assignment $I\colon V \to \{0, 1\}$
Objective:	maximize $w(I) = \sum_{C \in H\colon I(C)=1} w(C)$

Further Problems Our multiobjective algorithms rely upon approximations of the following multiobjective problems.

***k*-Objective Minimum Spanning Tree**

Notation:	k-MST
Instance:	$\mathbb{N}^k$-labeled undirected graph $G = (V, E, w)$
Solution:	spanning tree $T \subseteq E$
Objective:	minimize $w(T)$

***k*-Objective Shortest Path**

Notation:	k-SP
Instance:	$\mathbb{N}^k$-labeled undirected graph $G = (V, E, w)$ and $s, t \in V$
Solution:	path $P \subseteq E$ from s to t
Objective:	minimize $w(P)$

***k*-Objective Minimum Weight Perfect Matching**

Notation:	k-MinPM
Instance:	$\mathbb{N}^k$-labeled undirected graph $G = (V, E, w)$
Solution:	perfect matching $M \subseteq E$ of G
Objective:	minimize $w(M)$

***k*-Objective Maximum Weight Perfect Matching**

Notation:	k-MaxPM
Instance:	$\mathbb{N}^k$-labeled undirected graph $G = (V, E, w)$
Solution:	perfect matching $M \subseteq E$ of G
Objective:	maximize $w(M)$

Note that in the single-objective setting, minimum spanning trees, shortest paths, and perfect matchings can be computed in polynomial time, while already in the presence of two objectives, it is easy to construct families of problem instances such that there exists a number of incomparable solutions that is exponential in the length of the instance. Hence, for the above problems, an algorithm that computes the complete Pareto set in polynomial time does not exist.

Papadimitriou and Yannakakis [PY00] show that a multiobjective problem admits an FPTAS if its exact version is solvable in pseudo-polynomial time. Since the exact weight spanning tree problem was solved in pseudo-polynomial time by Barahona and Pulleyblank [BP87] and the exact weight shortest path problem is solvable in pseudo-polynomial time by dynamic programming [PY00], their multiobjective counterparts admit an FPTAS each. Furthermore, there exists a randomized pseudo-polynomial-time algorithm for the exact weight perfect matching problem if there exists a randomized polynomial-time algorithm for the exact perfect matching problem with unary weights. The latter problem can be reduced to the exact perfect matching problem with $(0, 1)$ weights [PY82], for which a randomized NC^2 algorithm exists [MVV87]. So overall, the following approximation schemes exist.

Theorem 3.2.1 ([PY00, BP87, PY82, MVV87]) *Let $k \geq 1$.*

1. *There exists an FPTAS for* k-MST.
2. *There exists an FPTAS for* k-SP.
3. *There exists an FPRAS for* k-MinPM.
4. *There exists an FPRAS for* k-MaxPM.

3.3 Previous Work and Contributions in this Part

Multiobjective Optimization For a general introduction to multiobjective optimization, we refer to the textbook by Ehrgott [Ehr05]. We further refer to Ehrgott and Gandibleux [EG02] who provide a detailed guide over the existing literature on multiobjective optimization up to that time. The paper by Papadimitriou and Yannakakis [PY00] is of particular importance for this thesis. First, it shows that for all multiobjective problems we consider in this thesis, even if the exact Pareto set consists of too many solutions, for arbitrary $\varepsilon > 0$ there exist $(1 + \varepsilon)$-approximate Pareto sets of polynomial cardinality in the length of the input instance.

So there might exist approximation algorithms. Moreover, we obtain Theorem 3.2.1 from their work, whose approximation algorithms we will often use in our algorithms. Third, they show how to obtain good approximation algorithms by solving a subproblem, where one either has to find a solution that approximately satisfies a given constraint, or to assert that no solution exists that satisfies the constraint. We will apply this approach in a later chapter. Finally we want to mention that general work on multiobjective optimization also considers further interpretations of "solving a multiobjective problem". For instance, one can aggregate the different objectives into a single objective by taking a weighted sum, which is then optimized, or one can also search for solutions that satisfy a given constraint. For a general comparison of different solution notions we refer to the work by Glaßer et al. [GRSW10a] and Fleszar et al. [FGL$^+$12].

Multiobjective Traveling Salesman Problems The previously best known approximation algorithms for the multiobjective minimum traveling salesman problem (with a multiobjective distance function that is metric in each component) is due to Manthey and Ram [MR09]. They deterministically approximate the Pareto set of Hamiltonian cycles by computing minimum spanning trees, doubling the edges, and taking shortcuts. In the multiobjective setting, this provides a $(2+\varepsilon)$-approximation for arbitrary small $\varepsilon > 0$.

For the symmetric and asymmetric maximization variants, the previously best known approximation algorithms are due to Manthey [Man12b], who gave a randomized $(2-\varepsilon)/3$-approximation for k-MaxSTSP and a randomized $(1-\varepsilon)/2$-approximation for k-MaxATSP for arbitrary $\varepsilon > 0$. Moreover, Manthey [Man12a] analyzed the deterministic approximability of these problems and showed that k-MaxATSP is deterministically $(1-\varepsilon)/(4k-2)$-approximable, and k-MaxSTSP is deterministically $(1-\varepsilon)/2k$-approximable, for $k \geq 2$ and $\varepsilon > 0$.

Improved Approximation Results We summarize the improvements on the approximability of the multiobjective variants of the traveling salesman problem that we show in the first part of this thesis in Table 3.1.

Problem	Deterministic Approximation	Randomized Approximation	Reference
Multigraph 2-MinTSP	$(2,2)$	$(3/2+\varepsilon,2)$, $(3/2,2+\varepsilon)$	4.3.2, 4.3.3
Multigraph 2-MinTSPP	$(2+\varepsilon,2+\varepsilon)$	$(3/2+\varepsilon,5/3+\varepsilon)$	4.4.3, 4.4.2
Multigraph 2-MinTSPPs	$(2+\varepsilon,2+\varepsilon)$	$(3/2+\varepsilon,2+\varepsilon)$	4.4.3, 4.4.2
Multigraph 2-MinTSPPst	$(2+\varepsilon,2+\varepsilon)$	$(2+\varepsilon,2+\varepsilon)$	4.4.3
k-MaxATSP	$(1-\varepsilon)/2$	$1/2$	6.5.2, 5.4.2
k-MaxSTSP	$(2-\varepsilon)/3$	$2/3$	6.5.2, 5.4.2

Table 3.1: Improved approximation ratios for TSP problems, where $\varepsilon > 0$ and $k \geq 1$.

We note that the multigraph problems cover the conventional problems considered by Manthey and Ram [MR09] as a special case. For the minimum traveling salesman and traveling salesman path problems studied in Chapter 4 we will further provide arguments that indicate the hardness of improving our approximation results.

In addition to the problems mentioned in Table 3.1, we obtain approximation results for further multiobjective problems. In Chapter 4 we translate existing approximation schemes for matching, shortest path, and minimum spanning tree to multigraphs. In Chapter 5 we additionally show that k-MaxSAT is $1/2$-approximable. In Chapter 6 we further consider the deterministic approximability of matching and cycle cover problems and obtain PTAS and PTAS-like approximation algorithms. Note that by Papadimitriou and Yannakakis [PY00], a

FPRAS for matching and hence for cycle cover exists. The algorithms we obtain have a weaker approximation ratio but work deterministically.

Vector Balancing Lemma In order to approximate k-MaxATSP and k-MaxSTSP, we solve the following problem. Suppose we are given a matrix of k-dimensional vectors of natural numbers, and we want to choose one vector per column such that the sum over the chosen vectors is small in each dimension. We provide a lemma that shows how to compute such a selection in polynomial time. In particular, we can guarantee that if the matrix consists of c rows, then in each dimension, the sum of our selection roughly equals a fraction of only $1/c$ of the sum of all vectors in the matrix. While this general balancing lemma provides us with a simple and elegant way to approximate k-MaxATSP and k-MaxSTSP, it solves a subproblem that frequently occurs in multiobjective optimization. Namely, it provides us with the opportunity to find a balanced selection of objects that have multidimensional weights. As another example, we show that our lemma provides a simple way to approximate k-MaxSAT.

Chapter 4

Minimum Traveling Salesman Problems

Introduction to Minimum Traveling Salesman Problems We first consider the approximability of minimum traveling salesman problems. In the single-objective setting, we are given an undirected, complete input graph with edge labels in $\mathbb{N}$, and we want to compute Hamiltonian cycles of minimum weight. We denote this problem by MinTSP.

Recall that a Hamiltonian cycle covers all vertices exactly once. Hence in the very general case of MinTSP, we want to find a roundtrip where no city is visited more than once. Johnson and Papadimitriou [JP85] state that the substantial majority of real-world traveling salesman problems, including all geometric versions, do not need this restriction. Moreover, this case does not admit a constant factor approximation algorithm, unless P = NP [SG76]. We concentrate on metric problem instances, where the distance function satisfies the triangle inequality. We will denote the corresponding restriction of MinTSP to problem instances with metric distance functions by MinΔTSP. It is easy to see that with a single objective, MinΔTSP is equivalent to the variant of MinTSP where multiple visits of cities are allowed.

For a long time, the best known approximation algorithm for the MinΔTSP was the simple tree-doubling method. For the input graph G, one proceeds as follows:

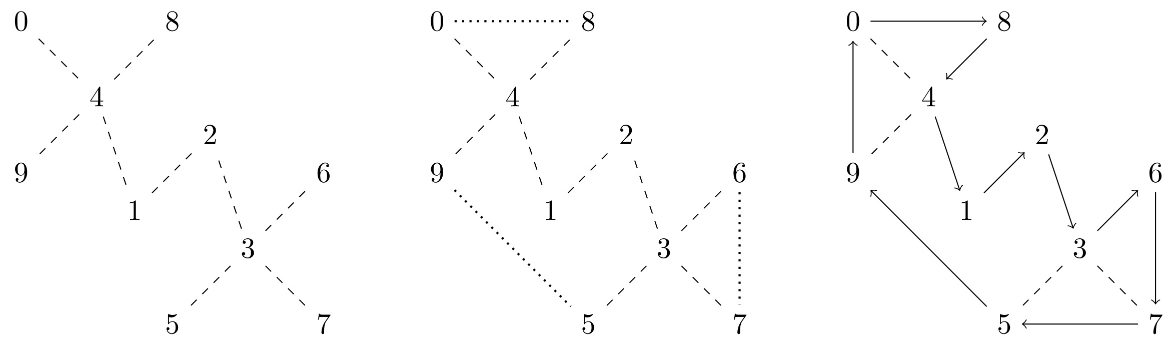
1. compute a minimum spanning tree T of G
2. double the edges of T to obtain an Eulerian multigraph H
3. compute an Eulerian tour in H and take shortcuts to obtain a Hamiltonian cycle C

Note that minimum spanning trees can be computed in polynomial time. The crucial observation is that every Hamiltonian cycle contains a spanning tree, and since T is minimal, it must have weight less than every Hamiltonian cycle. Taking shortcuts does not increase the weight, because the weight function is metric. Hence by the tree-doubling method, one obtains a 2-approximation.

The tree-doubling method was significantly improved by Christofides [Chr76], who observed that in order to obtain an Eulerian graph, it suffices to connect the edges of odd degree in T. On the input graph G, his approach works as follows:

1. compute a minimum spanning tree T of G
2. let U denote the vertices of odd degree in T
3. compute a minimum perfect matching M of U in G
4. combine the edges of T and M to obtain an Eulerian multigraph H
5. compute an Eulerian tour in H and take shortcuts to obtain a Hamiltonian cycle C

In addition to the arguments that we used for the tree-doubling method, he argued that each Hamiltonian cycle contains two distinct perfect matchings of U. Consequently, the minimum perfect matching M cannot weigh more than half of the weight of the Hamiltonian cycle, and we obtain a $3/2$-approximation. Figure 4.1 describes the execution of Christofides' algorithm on an example graph. We refer to standard textbooks on approximation algorithms [ACG+99, Vaz01] for a more detailed description of the single-objective tree-doubling method and Christofides' algorithm.



a) Minimum Spanning Tree b) Eulerian graph c) Hamiltonian Cycle

Figure 4.1: Example for the execution of Christofides' algorithm. Suppose we are given the complete input graph G with vertices $\{0, \dots, 9\}$, where the distance between two vertices is determined by their Euclidean distance. In the first step, we compute a minimum spanning tree T of G. T corresponds to the dashed edges. We obtain that the vertices $U = \{0, 5, 6, 7, 8, 9\}$ have odd degree in T. In the second step, we compute a minimum perfect matching M for the vertices in U. M corresponds to the dotted edges. Observe that $M \cup T$ connects all vertices, and all vertices have even degree, hence $M \cup T$ is an Eulerian graph. In the last step we compute a Hamiltonian cycle in $M \cup T$, where we follow an arbitrary closed walk and skip vertices that we have already visited. In the above example we started the walk at vertex 1 and skipped the vertices 3 and 4, since we visited them before already.

More recent work on the MinΔTSP improved the approximation ratio of Christofides' algorithm for special cases. Most prominently, Arora [Aro98] showed that there exists a PTAS for the Euclidean variant of MinΔTSP, where vertices are points in the plane, and where the distance between vertices is defined by their Euclidean distance. Another variant is studied by Papadimitriou and Yannakakis [PY93], who construct a $7/6$-approximation algorithm for the traveling salesman problem, where all distances are either 1 or 2. There also exist lower bounds on the approximability of the problem. Here, Papadimitriou and Vempala [PV06] showed that MinΔTSP cannot be approximated with a ratio better than $220/219$, unless $\text{P} = \text{NP}$. However, Christofides' algorithm is still the best known approximation of MinΔTSP.

Minimum Traveling Salesman Path Problems The metric traveling salesman problem motivates several path problems where for given cities, one has to find a shortest path that visits each city exactly once and that additionally starts or also ends in specified cities. We will use MinΔTSPP, MinΔTSPPs, and MinΔTSPPst to denote the metric traveling salesman path problems on undirected, complete, $\mathbb{N}$-labeled graphs with a metric distance function for 0, 1, and 2 specified end vertices, respectively. These problems were studied by Hoogeveen [Hoo91], who adapted Christofides' idea to the path setting and showed $3/2$-approximations for the problems MinΔTSPP and MinΔTSPPs, and a $5/3$-approximation for the problem MinΔTSPPst. The approximation algorithm for MinΔTSPPst was recently improved by An, Kleinberg and Shmoys [AKS12], who showed a $(1+\sqrt{5})/2$-approximation for the case of given start and end cities. For MinΔTSPP and MinΔTSPPs, Hoogeveens algorithms are still the best known approximation algorithms.

Multiple Objectives The multiobjective traveling salesman problem was first studied by Gupta and Warburton [GW86]. Angel, Bampis, and Gourvès [ABG04] give a $3/2$-approximation for the two-objective variant where the components of the distance vectors are either 1 or 2. Furthermore, Angel et al. [ABGM05] investigate the non-approximability of this problem. Ehrgott [Ehr00] studies the multiobjective traveling salesman problem and uses the l_1-norm (i.e., the sum of the components) to aggregate the weight functions into one. In general, the approximation ratio obtained by this approach is incomparable to the multi-component approximation ratios that we consider in this work (see Glaßer et al. [GRSW10a] and Fleszar et al. [FGL+12] for a detailed analysis of different solution and approximation notions of multiobjective problems). Manthey and Ram [MR09] apply a generalization of the tree-doubling method to the multiobjective traveling salesman problem, where the cost function is metric in each component. Their deterministic algorithm achieves an approximation ratio of $2 + \varepsilon$.

Motivation for our Work In contrast to tree-doubling, a straightforward adaption of Christofides' algorithm to multiple objectives fails. Recall that the crucial observation in the analysis of Christofides' algorithm is that a minimum weight Hamiltonian cycle can be split into two disjoint perfect matchings, and hence there must exist a perfect matching that weighs at most half as much as a minimum weight Hamiltonian cycle. Already in the presence of a second objective, it is possible that the two matchings we extract from a Hamiltonian cycle are incomparable, and so the above argumentation fails. So our goal is to either find approximation algorithms that are better than the simple tree-doubling method for multiple objectives, or to find arguments that show why the multiobjective case is harder than the single-objective case. Analogously, one can see that tree-doubling methods work for the multiobjective traveling salesman path problems as well. Here, the same problem arises when we switch to more than one objective, and we are also interested in better approximation algorithms and in hardness arguments.

Contributions We summarize the contributions of this chapter as follows.

1. *Generalized Realistic Models.* We propose a generalized definition for the multiobjective traveling salesman problem that is based on multigraphs and that allows multiple visits of cities. It captures typical real-world scenarios and it contains as a special case the conventional definition, the componentwise metric multiobjective traveling salesman problem [MR09, Ehr00].
2. *Improved Approximations.* We restrict ourselves to the case of two objectives. For the multigraph versions of the two-objective traveling salesman and of the two-objective traveling salesman path problems we provide deterministic and randomized approximation algorithms that improve the previously known results. We summarize our approximation results in Table 4.1.

Problem	Deterministic Approximation	Randomized Approximation	Reference
Multigraph 2-MinTSP	$(2, 2)$	$(3/2 + \varepsilon, 2)$, $(3/2, 2 + \varepsilon)$	4.3.2, 4.3.3
Multigraph 2-MinTSPP	$(2 + \varepsilon, 2 + \varepsilon)$	$(3/2 + \varepsilon, 5/3 + \varepsilon)$	4.4.3, 4.4.2
Multigraph 2-MinTSPPs	$(2 + \varepsilon, 2 + \varepsilon)$	$(3/2 + \varepsilon, 2 + \varepsilon)$	4.4.3, 4.4.2
Multigraph 2-MinTSPPst	$(2 + \varepsilon, 2 + \varepsilon)$	$(2 + \varepsilon, 2 + \varepsilon)$	4.4.3

Table 4.1: Approximation ratios shown in this chapter, where $\varepsilon > 0$.

3. *Approximation Schemes for Multigraph Problems.* Our approximation algorithms generalize Christofides' algorithm and the tree-doubling method to the multigraph setting. These algorithms, however, work with minimum spanning trees and matchings. We will first analyze the approximability of multiobjective minimum spanning tree, matching, and shortest path problems on multigraphs, and obtain approximation preserving reductions to the case of simple graphs. Together with Theorem 3.2.1 it follows that there exist approximation schemes for the multigraph problems.
4. *Lower Bound Arguments.* We present arguments that indicate the hardness of improving our approximation algorithms by showing approximation preserving reductions that allow us to translate approximation ratios for the two-objective multigraph minimum traveling salesman and traveling salesman path problems to the single-objective case. Table 4.2 summarizes our main arguments.

Problem	Approximation ratio of ...	... yields ratio of ...	... for problem	Reference
Multigraph 2-MinTSP	$(\frac{1+\sqrt{5}}{2}-\varepsilon_1, 2-\varepsilon_2)$	$\frac{1+\sqrt{5}}{2}-\varepsilon_1$	MinΔTSPPst	4.5.7
Multigraph 2-MinTSPP	$(\frac{1+\sqrt{5}}{2}-\varepsilon_1, \frac{3}{2}-\varepsilon_2)$	$\frac{1+\sqrt{5}}{2}-\varepsilon_1$	MinΔTSPPst	4.5.9
Multigraph 2-MinTSPP	$(2-\varepsilon_1, \frac{3}{2}-\varepsilon_2)$	$\frac{3}{2}-\varepsilon_2$	MinΔTSPPs	4.5.9
Multigraph 2-MinTSPPs	$(\frac{1+\sqrt{5}}{2}-\varepsilon_1, 2-\varepsilon_2)$	$\frac{1+\sqrt{5}}{2}-\varepsilon_1$	MinΔTSPPst	4.5.9

Table 4.2: Arguments that indicate the difficulty of improving the obtained randomized approximations, where $\varepsilon_1, \varepsilon_2 > 0$. The first row indicates that an approximation algorithm for Multigraph 2-MinTSP with an approximation ratio better than $(\frac{1+\sqrt{5}}{2}, 2)$ in both objectives yields an approximation algorithm for MinΔTSPPst with approximation ratio better than $\frac{1+\sqrt{5}}{2}$.

Organization of this Chapter Section 4.1 contains the preliminary definitions and notations used in this chapter. It also includes a definition of all optimization problems we approximate in this chapter. Furthermore, in this section we will motivate why we work on the more general multigraph model. In Section 4.2 we will then translate the existing approximation schemes for minimum spanning tree, matching, and shortest path to multigraphs. We will also introduce a combination of shortest path and matching problems, which will help us to deterministically approximate the two-objective minimum traveling salesman problem. We will present our approximation algorithms for the multiobjective traveling salesman problems in Section 4.3 and for the multiobjective traveling salesman path problems in Section 4.4. We will conclude the chapter with the lower bound arguments presented in Section 4.5 and a summary and discussion in Section 4.6.

4.1 Problem Definitions

Recall the definition of undirected graphs and multigraphs, and further recall the concepts of (closed, spanning) walks, and (Hamiltonian) paths and cycles. For a graph $G = (V, E)$ and $U \subseteq V$ with $|U|$ even, we define a *path matching of* U *in* G as a set P of $|U|/2$ paths in G such that every vertex in U is endpoint of exactly one path in P.

Main Problems Recall the definition of the generalized version of the minimum traveling salesman problem defined with vertex repetitions on multigraphs with $k \geq 1$ objectives.

***k*-Objective Minimum Traveling Salesman on Multigraphs**
Notation: Multigraph k-MinTSP
Instance: $\mathbb{N}^k$-labeled undirected multigraph $G = (V, E, w)$
Solution: closed spanning walk W
Objective: minimize $w(W)$

The conventional metric problem k-MinΔTSP is defined analogously, where the input is an $\mathbb{N}^k$-labeled undirected complete simple graph, whose weight function satisfies the triangle inequality in each component, and where we compute Hamiltonian cycles instead. If $k = 1$, then we omit the prefix k and use MinΔTSP.

Further recall the definition of the generalized versions of minimum traveling salesman path problems on multigraphs.

***k*-Objective Minimum Traveling Salesman Path on Multigraphs**
Notation: Multigraph k-MinTSPP
Instance: $\mathbb{N}^k$-labeled undirected multigraph $G = (V, E, w)$
Solution: spanning walk W
Objective: minimize $w(W)$

The problems Multigraph k-MinTSPPs (resp., Multigraph k-MinTSPPst) are defined analogously, where in addition to $G = (V, E, w)$, the input contains a vertex $s \in V$ (resp., two vertices $s, t \in V$), and the spanning walk must have endpoint s (resp., endpoints s and t).

We define the metric variants k-MinΔTSPP, k-MinΔTSPPs, and k-MinΔTSPPst analogously, where the input contains an $\mathbb{N}^k$-labeled undirected complete simple graph instead, whose weight function satisfies the triangle inequality in each component, and where we compute Hamiltonian paths. If $k = 1$, then we omit the prefix k and obtain MinΔTSPP, MinΔTSPPs, and MinΔTSPPst.

Justification of the Multigraph Model From a practical point of view, the conventional, componentwise metric definition of the multiobjective minimum traveling salesman problem is very restrictive. Consider a problem instance $G = (V, E, w)$ of 2-MinΔTSP. Hence $w\colon E \to \mathbb{N}^2$ is metric in each component. Moreover, for two vertices $u, v \in V$, suppose there exist two incomparable paths $P_1, P_2 \subseteq E$ from u to v in G with $w_1(P_1) < w_1(P_2)$ and $w_2(P_2) < w_2(P_1)$. If w is metric in each component, then there exists a path P from u to v with $w(P) \leq w(P_1)$ and $w(P) \leq w(P_2)$. However, for many real-world applications, this is not the case, and hence we switch to multigraphs and compute closed spanning walks. This reformulation captures the conventional definition as a special case but is more general.

Proposition 4.1.1 *Let $k \geq 1$ and $\alpha \in \mathbb{R}^k$ such that $\alpha_i \geq 1$ for all i.*

1. *If there exists an α-approximation algorithm for* Multigraph k-MinTSP*, then there exists an α-approximation algorithm for* k-MinΔTSP.
2. *If there exists an α-approximation algorithm for* Multigraph k-MinTSPP*, then there exists an α-approximation algorithm for* k-MinΔTSPP.
3. *If there exists an α-approximation algorithm for* Multigraph k-MinTSPPs*, then there exists an α-approximation algorithm for* k-MinΔTSPPs.
4. *If there exists an α-approximation algorithm for* Multigraph k-MinTSPPst*, then there exists an α-approximation algorithm for* k-MinΔTSPPst.

Proof We show the first item. The proof of the other items works analogously.

We interpret the complete input graph $G = (V, E, w)$ for k-MinΔTSP as a multigraph without multiple edges, call the α-approximation algorithm for Multigraph k-MinTSP on G, and obtain a set of closed spanning walks $\mathcal{W}$ of G. We output every $W \in \mathcal{W}$, where we take arbitrary shortcuts to obtain Hamiltonian cycles. Note that the last step is possible because the input graph is complete.

Let C be a Hamiltonian cycle of G. Then C can be interpreted as a closed spanning walk of G. Hence there exists a closed spanning walk $W \in \mathcal{W}$ with $w(W) \leq \alpha \cdot w(C)$. Taking shortcuts does not increase the weight, because the triangle inequality holds in every component, so the algorithm outputs a Hamiltonian cycle C' with $w(C') \leq \alpha \cdot w(C)$. □

As a consequence of Proposition 4.1.1, our approximation algorithms focus on Multigraph k-MinTSP, while we develop lower bound arguments on the approximation ratio for k-MinΔTSP.

Matchings, Spanning Trees, and Shortest Paths on Multigraphs The classical approximation algorithms for MinΔTSP work with minimum perfect matchings, minimum spanning trees and shortest path algorithms. We extend the problem definitions to multigraphs as follows.

***k*-Objective Minimum Spanning Tree on Multigraphs**

Notation:	Multigraph k-MST
Instance:	$\mathbb{N}^k$-labeled undirected multigraph $G = (V, E, w)$
Solution:	spanning tree $T \subseteq E$
Objective:	minimize $w(T)$

***k*-Objective Shortest Path on Multigraphs**

Notation:	Multigraph k-SP
Instance:	$\mathbb{N}^k$-labeled undirected multigraph $G = (V, E, w)$ and $s, t \in V$
Solution:	path $P \subseteq E$ from s to t
Objective:	minimize $w(P)$

***k*-Objective Minimum Weight Perfect Matching on Multigraphs**

Notation:	Multigraph k-MinPM
Instance:	$\mathbb{N}^k$-labeled undirected multigraph $G = (V, E, w)$
Solution:	perfect matching $M \subseteq E$ of G
Objective:	minimize $w(M)$

***k*-Objective Minimum Weight Perfect Matching Path on Multigraphs**

Notation:	Multigraph k-MinPMP
Instance:	$\mathbb{N}^k$-labeled undirected multigraph $G = (V, E, w)$ and $U \subseteq V$
Solution:	path matching $M \subseteq E$ of U in G
Objective:	minimize $w(M)$

4.2 Multigraph Approximation Schemes

Recall that by Theorem 3.2.1, there exist (randomized) approximation schemes for the multiobjective variants of the minimum spanning tree, minimum perfect matching, and shortest path problem on simple graphs. In this section we will translate this result to multigraphs, and further show that the minimum path matching problem on multigraphs also admits a randomized approximation scheme.

We will describe a reduction of each multigraph problem to the case of simple graphs, where for a given multigraph we construct a simple graph by splitting edges into separate parts and introducing new vertices. Observe that edges $(\{u,v\},i_1)$ and $(\{u,v\},i_2)$ of a given multigraph can be "simulated" in a simple graph by the edges $\{u,z_{i_1}\}$, $\{z_{i_1},v\}$, $\{u,z_{i_2}\}$, and $\{z_{i_2},v\}$, where z_{i_1} and z_{i_2} are new vertices. By a slight modification of this scheme (depending on the problem we consider) and an accordingly defined new weight function, we obtain reductions to the case of simple graphs. This shows the following result.

Theorem 4.2.1 *Let $k \geq 1$.*

1. *There exists an FPTAS for* Multigraph k-MST.
2. *There exists an FPTAS for* Multigraph k-SP.
3. *There exists an FPRAS for* Multigraph k-MinPM.
4. *There exists an FPRAS for* Multigraph k-MinPMP.

Proof We reduce the multigraph problems to their counterparts on simple graphs, for which approximation schemes are known by Theorem 3.2.1.

1. Multigraph Minimum Spanning Tree Let $G = (V,E,w)$ be an $\mathbb{N}^k$-labeled multigraph. We transform G to a simple graph G' by splitting each edge into three parts. 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 $w(e)$, while the remaining two edges are labeled with $(0,\ldots,0) \in \mathbb{N}^k$.

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

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

We extend f to subsets $E_1 \subseteq E$:

$$f(E_1) = \bigcup_{e\in E_1} f(e)$$

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.,

$$w(E_1) = w'(f(E_1)) \quad \text{and} \quad w'(E_1') = w(g(E_1')). \tag{4.1}$$

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,

$$E_1' \text{ is connected and covers } V \implies g(E_1') \text{ is connected and covers } V. \tag{4.2}$$

We describe the FPTAS for Multigraph k-MST on input $G = (V,E,w)$ and $\varepsilon > 0$: Transform G into G' as described above and run the FPTAS of Theorem 3.2.1 for k-MST on input G' and ε. We obtain a $(1+\varepsilon)$-approximation $\mathcal{A}$ of all minimum 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 spanning tree of G. 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 weight $w'(f(T)) = w(T)$. So $\mathcal{A}$ contains a spanning tree T' of G' such that $w'(T') \leq (1+\varepsilon)\cdot w(T)$. By (4.2), $g(T')$ is connected and by (4.1), $w(g(T')) = w'(T') \leq (1+\varepsilon)\cdot w(T)$. Hence the algorithm outputs a spanning tree of G with weight at most $(1+\varepsilon)\cdot w(T)$.

2. Multigraph Shortest Path We use exactly the same construction as in part 1. So recall the notions from part 1. The algorithm works as follows: On input of an $\mathbb{N}^k$-labeled multigraph (V, E, w), $s, t \in V$ and $\varepsilon > 0$, construct the simple graph $G' = (V \cup U, E', w')$ as in part 1, run the FPTAS of Theorem 3.2.1 for k-SP 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 w already noted in part 1.

3. Multigraph Minimum Weight Perfect Matching Let $G = (V, E, w)$ be an $\mathbb{N}^k$-labeled multigraph. We transform G to the same $\mathbb{N}^k$-labeled simple graph $G' = (V \cup U, E', w')$ as in the first part, only the weight function w' is defined differently. So recall U and E' from the first part. The label of the original edge $w(e)$ is put on both outer edges, while the edge in the middle is labeled with zero. More formally:

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

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-MinPM on input $G = (V, E, w)$ and $\varepsilon > 0$ works as follows: Transform G into G' as described above and run the FPRAS from Theorem 3.2.1 for k-MinPM on input G' and ε. If the FPRAS succeeds, we obtain a $(1 + \varepsilon)$-approximation $\mathcal{A}$ of all minimal perfect matchings of G'. For each $M' \in \mathcal{A}$, output $h(M')$.

The running time of the algorithm is polynomial in $|G|$ and $1/\varepsilon$, and the probability that the execution of the FPRAS succeeds is at least $1/2$.

So suppose that the called FPRAS is executed successfully. It remains to show that the returned edge sets are in fact perfect matchings and that they approximate the minimal perfect matchings of G.

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 $\{u, u_e\}, \{v_e, v\} \in M' \iff \{u_e, v_e\} \notin M'$. Consider an arbitrary vertex $v \in V$. Since $v \in V$ 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 $\{u, v\} \in h(M')$, hence v is matched exactly once, and $h(M')$ is a perfect matching of G'.

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

$$w'(M') = 2w(h(M')). \tag{4.3}$$

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

4. Multigraph Minimum Weight Path Matching We use the approximation schemes for multigraph minimum weight shortest path and multigraph minimum weight perfect matching.

Let $G = (V, E, w)$ be an $\mathbb{N}^k$-labeled multigraph, $U \subseteq V$ be a set of even cardinality and $\varepsilon > 0$. Let $\varepsilon' = \min\{\varepsilon/3, 1\}$.

We start with the construction of an $\mathbb{N}^k$-labeled multigraph $G' = (U, E', w')$ 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 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 and let $E' = \{(\{s,t\}, i) \mid s,t \in U, s \neq t \text{ and } 1 \leq i \leq m_{\{s,t\}}\}$ and $w'(\{s,t\}, i) = w(p_i^{\{s,t\}})$ for $(\{s,t\}, i) \in E'$. Now run the FPRAS for Multigraph k-MinPM on G' and ε'. If the FPRAS succeeds, we obtain an approximate Pareto set of perfect matchings $\mathcal{M}$ of G'. For each perfect matching $M \in \mathcal{M}$ of G', output $\{p_i^{\{s,t\}} \mid (\{s,t\}, i) \in M\}$.

The running time of the algorithm is polynomial in $|G|$ and $1/\varepsilon$, and the probability that the execution of the FPRAS succeeds is at least $1/2$.

So suppose that the called FPRAS is executed successfully. The returned sets are path matchings, since every vertex $s \in U$ is matched by exactly one edge in M, and hence is endpoint of 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 $w(p_i^{\{s,t\}}) \leq (1+\varepsilon')w(p)$. For $\tilde{P}$ being the set of these approximations $p_i^{\{s,t\}}$ for all paths $p \in P$, we obtain $w(\tilde{P}) \leq (1+\varepsilon')w(P)$. Furthermore, $\tilde{P}$ is a path matching and thus corresponds to a perfect matching M of G' with the same weight. This perfect matching is approximated by a perfect matching $\tilde{M}$ using the FPRAS for Multigraph k-MinPM. For the path matching $\tilde{P}'$ finally obtained from $\tilde{M}$, we have the inequality

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

□

4.3 Approximation of Multigraph 2-MinTSP

In the case of a single objective, we can compute minimum spanning trees and minimum perfect matchings in polynomial time. Hence, Christofides' algorithm beats the tree-doubling method in every respect. In the presence of multiple objectives, however, the two problems are not known to have equal approximability. While multiobjective minimum spanning trees can be approximated deterministically by an FPTAS, we only know that there exists a randomized approximation scheme for multiobjective minimum perfect matchings. However, Christofides' algorithm relies heavily upon a matching approximation. We will distinguish between deterministic, tree-doubling based algorithms, and randomized, Christofides-like algorithms that also work with randomized matching approximations.

4.3.1 Deterministic Approximation

Manthey and Ram [MR09] show how to adapt the tree-doubling method to k-MinΔTSP for $k \geq 2$. They first compute a $(1+\varepsilon)$-approximate Pareto set of spanning trees, and then double the edges of each tree to obtain a set of Eulerian graphs. Since the input graph is complete and componentwise metric, each Eulerian graph can be transformed into a Hamiltonian cycle by taking shortcuts. Moreover, since each Hamiltonian cycle contains a spanning tree, this results in a $(2+2\varepsilon)$-approximate Pareto set of Hamiltonian cycles.

We improve the result of Manthey and Ram for the case of two objectives. Observe that for each Hamiltonian cycle we can construct a spanning tree by removing the heaviest edge in a given objective. In the spanning tree approximation, we will choose $\varepsilon > 0$ small enough such that the error introduced is less than the weight we lost by removing the heavy edge. This enables us to remove the error ε in one objective. We show that instead of doubling the edges of the obtained spanning tree, we can combine it with edges from a second spanning tree, which will enable us to remove ε in a second objective. Hence, for the two-objective case, we will obtain a deterministic $(2, 2)$-approximation algorithm.

We will first describe our method to combine two different spanning trees. Here, for the second tree, we will have to switch to the notion of path matchings. For a tree T and two vertices u, v of T, let $\mathrm{p}_T(u, v)$ denote the (unique) path in T connecting u and v. We consider the algorithm `match`, which is described as follows.

```
Algorithm: match($U, T$)
  Input  : tree $T$ and subset $U$ of its vertices of even cardinality
  Output : path matching $P$ on $U$ in $T$
1 let $M \subseteq U \times U$ be some subdivision of $U$ into pairs
2 $P := \{\mathrm{p}_T(u_1, u_2) \mid (u_1, u_2) \in M\}$
3 while there are distinct but non-disjoint $\mathrm{p}_T(u, u'), \mathrm{p}_T(v, v') \in P$ do
4     $P := P \setminus \{\mathrm{p}_T(u, u'), \mathrm{p}_T(v, v')\}$
5     if $\mathrm{p}_T(u, v) \cap \mathrm{p}_T(u', v') = \emptyset$ then
6         $P := P \cup \{\mathrm{p}_T(u, v), \mathrm{p}_T(u', v')\}$
7     else
8         $P := P \cup \{\mathrm{p}_T(u, v'), \mathrm{p}_T(u', v)\}$    // note that $\mathrm{p}_T(u, v') \cap \mathrm{p}_T(u', v) = \emptyset$
9 return $P$
```

Lemma 4.3.1 *For $k \geq 1$, let $G = (V, E, w)$ be an $\mathbb{N}^k$-labeled multigraph and $T \subseteq E$ be a spanning tree of G. For every $U \subseteq V$ of even cardinality,* `match`(U, T) *finds in polynomial time a path matching M of U in T such that $w(M) \leq w(T)$.*

Proof Let m denote the number of edges of T, and $S(P, T) := \sum_{p \in P} |p|$ be the sum of the number of edges of all paths used in T for some path matching P. Note that at any time in the algorithm, the value of the variable P is a path matching on U in T. Clearly, $S(P, T) \leq m^2/2$, since there are at most m edges per path and at most $m/2$ distinct pairs of endpoints of paths. In every iteration, we redirect two paths, which reduces $S(P, T)$ by at least two. Hence, the algorithm terminates after at most $m^2/4$ iterations. Since all operations of the algorithm (comparison of two unique paths in a tree and set operations) are polynomially time-bounded, we obtain a polynomial-time algorithm.

After the termination of the algorithm, any two distinct $\mathrm{p}_T(u, v), \mathrm{p}_T(u', v') \in P$ are completely disjoint. We can now estimate the overall weight of P by $w(P) = \sum_{p \in P} w(p) \leq w(T)$. □

Theorem 4.3.2 Multigraph 2-MinTSP *is* $(2, 2)$*-approximable.*

Proof Let k-`Multigraph-MST`(G, ε) denote the FPTAS for k-objective minimum spanning tree on multigraphs from Theorem 4.2.1 on input of an $\mathbb{N}^k$-labeled multigraph G and approximation parameter $\varepsilon > 0$. We show that the algorithm `2-Multigraph-MinTSP` as described below is a $(2, 2)$-approximation algorithm for Multigraph 2-MinTSP.

We argue for correctness and polynomial runtime, and show that the algorithm has an approximation ratio of 2. Let $G = (V, E, w)$ be an arbitrary input and $n = |G|$.

```
Algorithm: 2-Multigraph-MinTSP($V, E, w$)
Input  : $\mathbb{N}^2$-labeled multigraph $G = (V, E, w)$
Output : closed spanning walks of $G$
1 call 2-Multigraph-MST($G$, $1/2|V|$) to obtain set of spanning trees $\mathcal{T}$
2 foreach $(T_1, T_2) \in \mathcal{T} \times \mathcal{T}$ do
3     $U := \{v \in V \mid \deg_{T_1}(v)$ is odd$\}$
4     $M :=$ match($U, T_2$)
5     output closed spanning walk of $G$ using the edges of $T_1$ and $M$
```

Correctness: We show that the algorithm works correctly. Fix an arbitrary iteration of the algorithm, where two spanning trees T_1, T_2 are considered. In the spanning tree T_1 of G, the set U of vertices of odd degree has even cardinality. By Lemma 4.3.1, M is a path matching of U in G, hence for each vertex v it holds that the degree of v in M is odd if and only if v is contained in U. Hence the graph formed by the edges of M and T is a spanning subgraph of G where each vertex has even degree, and this graph can trivially be transformed into a closed spanning walk of G. So in each iteration we obtain a closed spanning walk of G.

Runtime: 2-Multigraph-MST$(G, \frac{1}{2|V|})$ runs in time polynomial in $|G| + 2|V|$, hence the set $\mathcal{T}$ has polynomial cardinality. So there are polynomially many pairs of spanning trees, and we have polynomially many iterations of the loop. From Lemma 4.3.1 it follows that M can be computed in polynomial time. All remaining operations only take time polynomial in n, so we obtain that the overall running time is polynomial in n.

Approximation Ratio: Let W be an arbitrary closed spanning walk of G. We show that 2-Multigraph-MinTSP(V, E, w) outputs a closed spanning walk W' such that $w(W') \leq 2w(W)$.

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 $w_i(W_{p_i}) \geq \frac{1}{m} w_i(W)$. By removing W_{p_i} from W, the multiset of edges E_i thus obtained is connected and covers every vertex of V. Thus G contains spanning trees T_i' with no higher weights than E_i, which means that

$$w\left(T_1'\right) \leq \left(\left(1 - \frac{1}{m}\right) w_1\left(W\right), w_2\left(W\right)\right) \quad \text{and} \quad w\left(T_2'\right) \leq \left(w_1\left(W\right), \left(1 - \frac{1}{m}\right) w_2\left(W\right)\right).$$

The FPTAS for the minimum spanning tree 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

$$w\left(T_1\right) \leq \left(1 + \frac{1}{2m}\right) w\left(T_1'\right) \qquad \text{and} \qquad w\left(T_2\right) \leq \left(1 + \frac{1}{2m}\right) w\left(T_2'\right).$$

Fix the iteration of the loop with exactly this pair of trees and let M be as in the algorithm and W' denote the output in this iteration. From Lemma 4.3.1 we obtain $w(M) \leq w(T_2)$, hence

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

□

4.3.2 Randomized Approximation

We will now show how to improve the approximation ratio of Multigraph 2-MinTSP by using the matching algorithm from Theorem 3.2.1. This approach has the drawback of being randomized, which means that the algorithm we obtain may fail with probability at most $1/2$.

Theorem 4.3.3 *For every $\varepsilon > 0$, Multigraph 2-MinTSP is randomized $(3/2 + \varepsilon, 2)$-approximable and randomized $(3/2, 2 + \varepsilon)$-approximable.*

Proof We show that the algorithm `Min2TSPApproxRand`$_\varepsilon$ as described below is a randomized $(3/2 + \varepsilon, 2)$-approximation algorithm and a randomized $(3/2, 2 + \varepsilon)$-approximation algorithm for Multigraph 2-MinTSP.

```
Algorithm: Min2TSPApproxRand_ε(V, E, w)
  Input : $\mathbb{N}^2$-labeled multigraph $G = (V, E, w)$
  Output: closed spanning walks of $G$
1 $m := |V|$, $\varepsilon_1 := \varepsilon/m^2$, $\varepsilon_2 := \varepsilon/2m$
2 compute $\varepsilon_1$-approximation $\mathcal{P}$ of minimum spanning trees for $G$ using the FPTAS from
  Theorem 4.2.1
3 foreach $T \in \mathcal{P}$ do
4     $U := \{v \in V \mid \deg_T(v) \text{ is odd}\}$
5     compute $\varepsilon_2$-approximation $\mathcal{A}$ of minimum weight path matchings for $U$ in $G$ using a
      polynomially amplified version of the FPRAS from Theorem 4.2.1
6     foreach $M \in \mathcal{A}$ do
7         output closed spanning walk using the edges of $T$ and $M$
```

We will now argue for correctness, polynomial runtime, success probability, and the claimed approximation ratio. We consider a fixed $1 > \varepsilon > 0$. Let $G = (V, E, w)$ be an arbitrary input graph, and let $m = |V|$ and $n = |G|$. We assume that $m \geq 2$.

Correctness: In every spanning tree T of G, the set U of vertices of odd degree has even cardinality. Moreover, if M is a path matching of U in G, then $v \in V$ has odd degree in M if and only if $v \in U$. Together we obtain that the graph formed by the edges of M and T is a spanning subgraph of G where each vertex has even degree, which can trivially be transformed into a closed spanning walk of G. Hence whenever all randomized parts of the algorithm succeed, we output a closed spanning walk of G.

Runtime: Observe that the FPTAS for minimum spanning tree and the polynomially amplified version of the FPRAS for minimum path matching run in time polynomial in $n + m^2/\varepsilon$ and $n + 2m/\varepsilon$, respectively. Since ε is constant and $m \leq n$, this is polynomial in n, and hence the sets $\mathcal{P}$ and $\mathcal{A}$ (for each $T \in \mathcal{P}$) have cardinality polynomial in n. Hence the number of nested iterations is polynomial in n. In each nested iteration, all remaining operations only take time polynomial in n, so we obtain that the overall running time is polynomial in n.

Success Probability: The only randomized parts of the algorithm are the calls of the amplified version of the FPRAS from Theorem 4.2.1 in line 5. Let p be the polynomial that bounds the runtime of the FPTAS for minimum spanning tree for ε fixed. Hence there are at most $p(n)$ calls of the amplified version of the FPRAS for minimum path matching. We use an amplified version of the FPRAS where the original FPRAS is called $p(n)$ times, hence the amplified version of the FPRAS has success probability of at least $1 - 1/2^{p(n)}$, and by Lemma 3.1.2, the probability that all calls of the amplified FPRAS and hence the overall algorithm succeed is at least $(1 - 1/2^{p(n)})^{p(n)} \geq \frac{1}{2}$.

Approximation Ratio: Suppose that the algorithm succeeds and consider an arbitrary closed spanning walk W of G. We show that the algorithm outputs a $(3/2 + \varepsilon, 2)$-approximation and a $(3/2, 2 + \varepsilon)$-approximation of W.

We split W into contiguous subwalks $W_1, \dots, W_m$ 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. Then, there is some $1 \leq r \leq m$ such that $w_2(W_r) \geq \frac{1}{m} w_2(W)$. By removing W_r from W, the multiset of edges E' thus obtained is connected and covers every vertex of V. Thus, G has a spanning tree T' with no higher weight than E', which means that

$$w\left(T'\right) \quad \leq \quad \left(w_1\left(W\right), \frac{m-1}{m} w_2\left(W\right)\right).$$

By $\varepsilon_1 = \varepsilon/m^2$, in line 2 we find a spanning tree T_1 with weight

$$w(T_1) \quad \leq \quad \left(1 + \frac{\varepsilon}{m^2}\right)\left(w_1(W), \frac{m-1}{m} w_2(W)\right).$$

By a symmetric argumentation, in line 2 we find a spanning tree T_2 with weight

$$w\left(T_2\right) \quad \leq \quad \left(1 + \frac{\varepsilon}{m^2}\right)\left(\frac{m-1}{m} w_1\left(W\right), w_2\left(W\right)\right).$$

Let $U_1 \subseteq V$ be the vertices of odd degree in T_1 and note that U_1 has even cardinality. From W we can easily construct two path matchings M_1 and M_2 of U_1 in G such that $w(M_1) + w(M_2) \leq w(W)$: For each $u \in U_1$, fix the first occurrence in W, and cut W at each of those $|U_1|$ positions to obtain $|U_1|$ subwalks $S_1, \dots, S_{|U_1|}$. For each i, we remove any cycles from S_i and obtain a path P_i with $w(P_i) \leq w(S_i)$. Then, both $M_1 = \{P_i \mid i \text{ even}\}$ and $M_2 = \{P_j \mid j \text{ odd}\}$ are path matchings of U_1 in G. Hence, there is some path matching M' of U_1 in G such that $w(M') \leq (\frac{1}{2} w_1(W), w_2(W))$.

By $\varepsilon_2 = \varepsilon/2m$, in the iteration of line 3 where $T = T_1$ we find some approximate minimum path matching M_1 of U_1 in G such that

$$w\left(M_1\right) \leq \left(1 + \frac{\varepsilon}{2m}\right) w\left(M'\right) \leq \left(1 + \frac{\varepsilon}{2m}\right)\left(\frac{1}{2} w_1\left(W\right), w_2\left(W\right)\right).$$

Again using symmetric arguments, in the iteration of line 3 where $T = T_2$ we find some approximate minimum path matching M_2 of U_2 in G such that

$$w\left(M_2\right) \quad \leq \quad \left(1 + \frac{\varepsilon}{2m}\right)\left(\frac{1}{2} w_1\left(W\right), w_2\left(W\right)\right).$$

By combining T_1 and M_1 we obtain a spanning walk W_1' such that

$$\begin{aligned} w_1\left(W_1'\right) \leq w_1\left(T_1\right) + w_1\left(M_1\right) &\leq \left(1 + \frac{\varepsilon}{m^2}\right) w_1\left(W\right) + \left(1 + \frac{\varepsilon}{2m}\right) \frac{1}{2} w_1\left(W\right) \\ &= \left(\frac{3}{2} + \varepsilon\left(\frac{1}{m^2} + \frac{1}{4m}\right)\right) w_1\left(W\right) \leq \left(\frac{3}{2} + \varepsilon\right) w_1\left(W\right) \end{aligned}$$

and

$$\begin{aligned} w_2\left(W_1'\right) \le w_2\left(T_1\right) + w_2\left(M_1\right) &\le \left(\left(1+\frac{\varepsilon}{m^2}\right)\frac{m-1}{m} + \left(1+\frac{\varepsilon}{2m}\right)\right) w_2\left(W\right) \\ &\le \left(\left(1+\frac{1}{m^2}\right)\frac{m-1}{m} + 1 + \frac{1}{2m}\right) w_2\left(W\right) \\ &= \left(2 + \frac{2m-m^2-2}{2m^3}\right) w_2\left(W\right) \le 2w_2\left(W\right). \end{aligned}$$

This shows the first part of the theorem. For the second part, we combine T_2 and M_2, which results in a spanning walk W_2' such that:

$$\begin{aligned} w_1\left(W_2'\right) \le w_1\left(T_2\right) + w_1\left(M_2\right) &\le \left(1+\frac{\varepsilon}{m^2}\right)\frac{m-1}{m} w_1\left(W\right) + \left(1+\frac{\varepsilon}{2m}\right)\frac{1}{2} w_1\left(W\right) \\ &\le \left(1+\frac{1}{m^2}\right)\frac{m-1}{m} w_1\left(W\right) + \left(1+\frac{1}{2m}\right)\frac{1}{2} w_1\left(W\right) \\ &= \left(\frac{3}{2} + \frac{4m-3m^2-4}{4m^3}\right) w_1(W) \le \frac{3}{2} w_1\left(W\right) \end{aligned}$$

and

$$\begin{aligned} w_2\left(W_2'\right) \le w_2\left(T_2\right) + w_2\left(M_2\right) &\le \left(1+\frac{\varepsilon}{m^2}\right) w_2\left(W\right) + \left(1+\frac{\varepsilon}{2m}\right) w_2\left(W\right) \\ &\le \left(1+\frac{\varepsilon}{4}\right) w_2\left(W\right) + \left(1+\frac{\varepsilon}{4}\right) w_2\left(W\right) \le \left(2+\varepsilon\right) w_2\left(W\right). \end{aligned}$$

□

4.4 Approximation of Multigraph 2-MinTSPP

Recall the definition of MULTIGRAPH k-MINTSPP, MULTIGRAPH k-MINTSPPS, and MULTIGRAPH k-MINTSPPST. We first show the following results.

1. MULTIGRAPH 2-MINTSPP is randomized $(3/2+\varepsilon, 5/3+\varepsilon)$-approximable.
2. MULTIGRAPH 2-MINTSPPS is randomized $(3/2+\varepsilon, 2+\varepsilon)$-approximable.

We further use a tree-doubling technique to show that for arbitrary k, the problems MULTIGRAPH k-MINTSPP, MULTIGRAPH k-MINTSPPS, and MULTIGRAPH k-MINTSPPST are deterministically $(2+\varepsilon)$-approximable.

4.4.1 Randomized Approximation

We will use the following lemma.

Lemma 4.4.1 *Let $G=(V,E,w)$ be a $\mathbb{N}^2$-labeled multigraph, let W be a spanning walk of G, and let $U \subseteq V$ such that $U \neq \emptyset$.*

1. *If $|U|$ is odd, then there exist a vertex $s \in U$ and a path matching M of $U-\{s\}$ in G such that $w_1(M) \le \frac{1}{2}w_1(W)$ and $w_2(M) \le w_2(W)$.*
2. *If $|U|$ is even, then there exist distinct vertices $s,t \in U$ and a path matching M of $U-\{s,t\}$ in G such that $w_1(M) \le \frac{1}{2}w_1(W)$ and $w_2(M) \le \frac{2}{3}w_2(W)$.*

Proof The lemma is obvious for $|U| < 3$, so assume $|U| \geq 3$. Furthermore we assume that the walk is not closed (otherwise, we remove the last edge of W, which only decreases the weight further).

Let v_1 and v_r denote the start and the end vertex of W, and let $(v_2, \dots, v_{r-1})$ denote the vertices of $U - \{v_1, v_r\}$ in the order of their first appearance in W. The first appearance of each v_i, $2 \leq i \leq r-1$, partitions W into a set of $r-1$ subwalks $\{W_1, W_2, \dots, W_{r-1}\}$, where W_i connects v_i and v_{i+1}. We extract a set of paths $P = \{P_1, P_2, \dots, P_{r-1}\}$ by removing cycles in each W_i. It holds that $w(P) \leq w(W)$ and P_i connects v_i and v_{i+1}. Define the following distinct sets that partition P.

$$M_{\text{odd}} := \{P_i \in P \mid i \text{ is odd}\}$$
$$M_{\text{even}} := \{P_i \in P \mid i \text{ is even}\}$$

We will now show both items of the lemma.

1. Suppose $|U|$ is odd. Then, M_{odd} is a path matching on $U - \{v_r\}$, and M_{even} is a path matching on $U - \{v_1\}$. Choose the path matching with the lower weight in w_1. From $M_{\text{even}} \cup M_{\text{odd}} = P$ and $w(P) \leq w(W)$, the first part of the lemma follows.
2. Suppose $|U|$ is even. Then, M_{odd} is a path matching on U and M_{even} is a path matching on $U - \{v_1, v_r\}$. We show this part by contradiction. Hence assume that the second part of the lemma does not hold, which means that for all distinct $s, t \in U$ and every path matching M of $U - \{s, t\}$ it holds that

$$w_2(M) \leq \tfrac{2}{3} w_2(W) \implies w_1(M) > \tfrac{1}{2} w_1(W). \tag{4.4}$$

This also holds for every path matching of U, since otherwise, by removing an arbitrary path, we obtain a path matching that leaves two vertices unmatched and that contradicts (4.4). We have the following cases.

Case 1: $w_2(M_{\text{odd}}) \leq \frac{2}{3} w_2(W)$ and $w_2(M_{\text{even}}) \leq \frac{2}{3} w_2(W)$.
From $w(M_{\text{even}}) + w(M_{\text{odd}}) = w(P) \leq w(W)$ it follows that $w_1(M_{\text{odd}}) \leq \frac{1}{2} w_1(W)$ or $w_1(M_{\text{even}}) \leq \frac{1}{2} w_1(W)$. So M_{odd} or M_{even} contradicts (4.4).

Case 2: $w_2(M_{\text{odd}}) > \frac{2}{3} w_2(W)$.
Since $w(M_{\text{even}}) + w(M_{\text{odd}}) = w(P) \leq w(W)$, we have

$$w_2(M_{\text{even}}) \leq w_2(W) - w_2(M_{\text{odd}}) < \tfrac{1}{3} w_2(W). \tag{4.5}$$

For every $1 \leq k < r$ we partition $P - \{P_k\}$ into the following sets:

$$L_k := \{P_i \mid 1 \leq i < k\} \qquad\qquad R_k := \{P_i \mid k < i < r\}$$

Consider the largest odd k such that $w_2(L_k \cap M_{\text{odd}}) \leq \frac{1}{2} w_2(M_{\text{odd}})$. From $P_k \in M_{\text{odd}}$, $P_k \notin L_k \cup R_k$, and the maximality of k, it follows that $w_2(R_k \cap M_{\text{odd}}) \leq \frac{1}{2} w_2(M_{\text{odd}})$. We define the following sets:

$$M_1 := (L_k \cap M_{\text{odd}}) \cup (R_k \cap M_{\text{even}})$$
$$M_2 := (L_k \cap M_{\text{even}}) \cup (R_k \cap M_{\text{odd}})$$

Observe that M_1 is a path matching on $U - \{v_k, v_r\}$ and M_2 is a path matching on $U - \{v_1, v_{k+1}\}$. We argue that either M_1 or M_2 contradicts (4.4).

Let us estimate the weights in the second component of M_1 and M_2:

$$\begin{aligned} w_2(M_1) &= w_2(L_k \cap M_{\text{odd}}) + w_2(R_k \cap M_{\text{even}}) & w_2(M_2) &= w_2(L_k \cap M_{\text{even}}) + w_2(R_k \cap M_{\text{odd}}) \\ &\leq \tfrac{1}{2} w_2(M_{\text{odd}}) + w_2(M_{\text{even}}) & &\leq w_2(M_{\text{even}}) + \tfrac{1}{2} w_2(M_{\text{odd}}) \end{aligned}$$

By (4.5) we know that $w_2(M_{\text{even}}) < \frac{1}{3} w_2(W)$ and thus for all $i \in \{1, 2\}$ we obtain

$$\begin{aligned} w_2(M_i) \leq \tfrac{1}{2} w_2(M_{\text{odd}}) + w_2(M_{\text{even}}) &= \tfrac{1}{2}(w_2(P) - w_2(M_{\text{even}})) + w_2(M_{\text{even}}) \\ &= \tfrac{1}{2}(w_2(P) + w_2(M_{\text{even}})) < \tfrac{1}{2}(w_2(W) + \tfrac{1}{3} w_2(W)) = \tfrac{2}{3} w_2(W). \end{aligned}$$

Note that M_1 and M_2 are disjoint. Therefore, $w(M_1) + w(M_2) \leq w(P) \leq w(W)$ and hence $w_1(M_1) \leq \frac{1}{2} w_1(W)$ or $w_1(M_2) \leq \frac{1}{2} w_1(W)$. So M_1 or M_2 contradicts (4.4).

Case 3: $w_2(M_{\text{even}}) > \frac{2}{3} w_2(W)$.
This case is very similar to the second, so we concentrate on the differences. We get $w_2(M_{\text{odd}}) < \frac{1}{3} w_2(W)$ and define L_k and R_k in the same way as in the second case. Consider now the largest *even* k such that $w_2(L_k \cap M_{\text{even}}) \leq \frac{1}{2} w_2(M_{\text{even}})$. From $P_k \in M_{\text{even}}$, $P_k \notin L_k \cup R_k$, and the maximality of k, it follows that $w_2(R_k \cap M_{\text{even}}) \leq \frac{1}{2} w_2(M_{\text{even}})$. We now show that either the path matching $M_1 := (L_k \cap M_{\text{odd}}) \cup (R_k \cap M_{\text{even}})$ on $U - \{v_{k+1}, v_r\}$ or the path matching $M_2 := (L_k \cap M_{\text{even}}) \cup (R_k \cap M_{\text{odd}})$ on $U - \{v_1, v_k\}$ contradicts (4.4). For all $i \in \{1, 2\}$ we similarly get $w_2(M_i) \leq w_2(M_{\text{odd}}) + \frac{1}{2} w_2(M_{\text{even}})$ and using $w_2(M_{\text{odd}}) < \frac{1}{3} w_2(W)$ we obtain $w_2(M_i) < \frac{2}{3} w_2(W)$. Since again $w_1(M_1) \leq \frac{1}{2} w_1(W)$ or $w_1(M_2) \leq \frac{1}{2} w_1(W)$ holds, M_1 or M_2 contradicts (4.4).

□

With Lemma 4.4.1, we can now prove that there exist the following approximations.

Theorem 4.4.2 *Let $\varepsilon > 0$.*

1. Multigraph 2-MinTSPP *is randomized* $(3/2 + \varepsilon, 5/3 + \varepsilon)$*-approximable.*
2. Multigraph 2-MinTSPPs *is randomized* $(3/2 + \varepsilon, 2 + \varepsilon)$*-approximable.*

Proof We consider the following algorithm. Recall that $\triangle$ denotes the symmetric difference operator on sets.

```
Algorithm: ApproxMin2TSPP$_\varepsilon$($V, E, w, S$)
  Input  : $\mathbb{N}^2$-labeled multigraph $G = (V, E, w)$, and $S \subseteq V$ with $|S| \leq 1$
  Output : spanning walks of $G$ (each using the startpoint in $S$, if $S \neq \emptyset$)
1 compute $\varepsilon/2$-approximation $\mathcal{P}$ of spanning trees for $G$ using FPTAS by Theorem 4.2.1
2 foreach $T \in \mathcal{P}$ do
3     $U := \{v \in V \mid \deg_T(v)$ is odd$\} \triangle S$
4     foreach $R \subseteq U$ such that $1 \leq |R| \leq 2$ and $|U - R|$ is even do
5         compute $\varepsilon/2$-approximation $\mathcal{A}$ of minimum weight path matchings for $U - R$ in $G$
          using a polynomially amplified version of the FPRAS from Theorem 4.2.1
6         foreach $M \in \mathcal{A}$ do
7             output spanning walk using edges of $T$ and $M$, starting at $s \in S$ if $S \neq \emptyset$
```

We argue for correctness, polynomial runtime, success probability, and the claimed approximation ratio. We consider a fixed $1 > \varepsilon > 0$. Let $G = (V, E, w)$ be an arbitrary input graph, and let $m = |V|$ and $n = |G|$. We assume that $m \geq 2$. Furthermore, let $S \subseteq V$ such that $|S| \leq 1$.

Correctness: Consider an arbitrary spanning tree T of G. Choose U for T as in the algorithm, and consider an arbitrary set R chosen in line 4. The set $U - R$ has even cardinality, so let M be an arbitrary path matching for $U - R$ in G. Now consider the graph W consisting of all edges from T and M. Observe that W is connected. We have the following cases.

- $S = \emptyset$. Then, U has even cardinality, hence $|R| = 2$, and so in W, exactly two vertices have odd degree, hence W corresponds to a spanning walk in G.
- $S = \{s\}$. Then, U has odd cardinality, hence $|R| = 1$. Let $R = \{t\}$. We further distinguish the following cases.
  - $s \neq t$. If s has even degree in T, then $s \in U$, hence s is matched in M, and so the degree of s in W is odd. If s has odd degree in T, then $s \notin U$, hence the degree of s remains odd in W. In both cases, the vertex t is the only other vertex of odd degree that is not matched by M, hence W corresponds to a spanning walk connecting s with t and thus starting at s.
  - $s = t$. Hence, $s \in R \subseteq U$, so s must have even degree in T. In this case, $U - R$ is the set of all vertices of odd degree in T, which are then matched in M. Hence W is a closed spanning walk, which we can shorten to a spanning walk starting at s.

Hence, in each case, we output spanning walks of G, and if $S = \{s\}$, then each spanning walk starts at s.

Runtime: Since we assume $\varepsilon > 0$ to be constant, the runtime of the FPTAS for minimum spanning tree and the polynomially amplified version of the FPRAS for minimum path matching is polynomial in n. Hence the sets $\mathcal{P}$ and $\mathcal{A}$ (in each iteration) have polynomially bounded cardinality. Moreover, there are at most n^2 sets $R \subseteq U$ such that $|R| \leq 2$. This means that the number of nested iterations of the algorithm is polynomially bounded in n. Observe that in each iteration, every step of the algorithm can be carried out in time polynomial in n. Hence the overall algorithm has runtime polynomial in n.

Success Probability: The only randomized parts of the algorithm are the calls of the amplified version of the FPRAS from Theorem 4.2.1 in line 5. Let p be the polynomial that bounds the runtime of the FPTAS for minimum spanning tree for ε fixed, and note that the loop in line 4 is iterated at most n^2 often. Hence there are at most $p(n) \cdot n^2$ calls of the amplified version of the FPRAS for minimum path matching. We use an amplified version of the FPRAS where the original FPRAS is called $p(n) \cdot n^2$ times, hence the amplified version of the FPRAS has success probability of at least $1 - \frac{1}{2^{p(n)\cdot n^2}}$, and by Lemma 3.1.2, it follows that the probability that all calls of the amplified FPRAS and hence the overall algorithm succeed is at least $(1 - \frac{1}{2^{p(n)\cdot n^2}})^{p(n)\cdot n^2} \geq \frac{1}{2}$.

Approximation Ratio: We first consider the case where $S = \emptyset$. Suppose that the algorithm succeeds and consider an arbitrary spanning walk W of G. We show that the algorithm outputs a $(3/2 + \varepsilon, 5/3 + \varepsilon)$-approximation of W.

The spanning walk W can be transformed into a spanning tree of at most the same weight by cutting off some edges. Hence there exists an approximation $T' \in \mathcal{P}$ such that $w(T') \leq (1 + \frac{\varepsilon}{2})w(W)$. Fix the iteration in line 2 with $T = T'$, hence $w(T) \leq (1 + \frac{\varepsilon}{2})w(W)$.

As in the algorithm, let U denote the set of vertices of odd degree in T. Clearly, U is non-empty and has even cardinality. By Lemma 4.4.1, there exist distinct vertices $s, t \in U$ and a path matching M' of $U - \{s,t\}$ in G with $w_1(M') \leq \frac{1}{2}w_1(W)$ and $w_2(M') \leq \frac{2}{3}w_2(W)$. Fix the iteration in line 4 where $R = \{s, t\}$.

The FPRAS for path matching finds a path matching M'' of $U - R$ in G such that $w(M'') \leq (1+\frac{\varepsilon}{2})w(M')$. Fix the iteration in line 6 where $M = M''$, hence we have $w_1(M) \leq (1+\frac{\varepsilon}{2})\frac{1}{2}w_1(W)$ and $w_2(M) \leq (1+\frac{\varepsilon}{2})\frac{2}{3}w_2(W)$.

Let W' denote a spanning walk using the edges of T and M. From $w(W') = w(T) + w(M)$ we obtain

$$\begin{aligned} w_1(W') &\leq \left(1+\tfrac{\varepsilon}{2}\right) w_1(W) + \left(1+\tfrac{\varepsilon}{2}\right)\left(\tfrac{1}{2}w_1(W)\right) \leq (\tfrac{3}{2}+\varepsilon)w_1(W) \text{ and} \\ w_2(W') &\leq \left(1+\tfrac{\varepsilon}{2}\right) w_2(W) + \left(1+\tfrac{\varepsilon}{2}\right)\left(\tfrac{2}{3}w_2(W)\right) \leq (\tfrac{5}{3}+\varepsilon)w_2(W). \end{aligned}$$

Now consider the case where $S = \{s\}$ for some $s \in V$. We only point out the differences to the above estimation. We additionally assume that W starts at s. Again we find an $(1+\frac{\varepsilon}{2})$-approximate spanning tree T' for W and fix the iteration with $T = T'$. The set U obtained in the next step has odd cardinality, so Lemma 4.4.1 only assures that there exists a vertex $t \in U$ and a path matching M' of $U - \{t\}$ in G with $w_1(M') \leq \frac{1}{2}w_1(W)$ and $w_2(M') \leq w_2(W)$. Again we fix the iteration with $R = \{t\}$. In this iteration we find a path matching M'' such that $w(M'') \leq (1+\frac{\varepsilon}{2})w(M')$. In the iteration where $M = M''$ we let W' denote the spanning walk obtained and get

$$\begin{aligned} w_1(W') &\leq \left(1+\tfrac{\varepsilon}{2}\right) w_1(W) + \left(1+\tfrac{\varepsilon}{2}\right)\left(\tfrac{1}{2}w_1(W)\right) \leq (\tfrac{3}{2}+\varepsilon)w_1(W) \text{ and} \\ w_2(W') &\leq \left(1+\tfrac{\varepsilon}{2}\right) w_2(W) + \left(1+\tfrac{\varepsilon}{2}\right) w_2(W) = (2+\varepsilon)w_2(W). \end{aligned}$$

□

4.4.2 Deterministic Approximation

We use a tree-doubling method to obtain deterministic approximations for the multiobjective traveling salesman path problems on multigraphs.

Theorem 4.4.3 *Let $k \geq 1$ and $\varepsilon > 0$.*

1. Multigraph k-MinTSPP *is $(2+\varepsilon)$-approximable.*
2. Multigraph k-MinTSPPs *is $(2+\varepsilon)$-approximable.*
3. Multigraph k-MinTSPPst *is $(2+\varepsilon)$-approximable.*

Proof We first show that tree doubling deterministically finds a $(2+\varepsilon)$-approximation for Multigraph k-MinTSPPst.

Let $G = (V, E, w)$ be an arbitrary $\mathbb{N}^k$-labeled multigraph, $s, t \in V$ with $s \neq t$ and $\varepsilon > 0$. First, compute an $\frac{\varepsilon}{2}$-approximation $\mathcal{A}$ of minimum spanning trees for G using the FPTAS from Theorem 4.2.1. For each tree $T \in \mathcal{A}$ we double each edge in T and then delete the unique path from s to t once. Clearly, we obtain a connected multigraph whose vertices have even degree except for s and t. Therefore, for each spanning tree T we can find a spanning walk W' from s to t with weight $w(W') \leq 2w(T)$.

Fix any arbitrary spanning walk W from s to t. Since W contains a spanning tree, there is a spanning tree $T \in \mathcal{A}$ such that $w(T) \leq (1+\frac{\varepsilon}{2})w(W)$. By the tree doubling method we get a spanning walk W' from s to t with $w(W') \leq 2w(T) \leq (2+\varepsilon)w(W)$.

It remains to argue for Multigraph k-MinTSPP and Multigraph k-MinTSPPs. These problems can be reduced to Multigraph k-MinTSPPst by iterating over all vertices $t \in V$ and $s, t \in V$, respectively. □

4.5 Lower Bound Arguments

We will now discuss arguments that indicate the hardness of finding better approximations for the two-objective traveling salesman and traveling salesman path problems. By Proposition 4.1.1, it suffices to consider such arguments for the conventional two-objective problems defined on simple graphs with componentwise metric weight functions, because their approximability reduces to the general case of multigraphs.

4.5.1 Lower Bound Arguments for TSP

We first show that MinΔTSPPst can be reduced to 2-MinΔTSP in an approximation preserving way. Given a complete metric input graph for MinΔTSPPst with start and end vertex, we will construct an instance of 2-MinΔTSP that contains two copies of the input graph, which are connected by two paths that consist of sufficiently many additional edges, such that the first path connects the copies of the start vertex, and the second path connects the copies of the end vertex. Our construction introduces a second weight function and makes sure that every Hamiltonian cycle in the new graph contains a Hamiltonian path in the first and a Hamiltonian path in the second copy. We will use the path with the lower weight to obtain a good solution to the original MinΔTSPPst instance.

Theorem 4.5.1 *For every $\alpha > 1$ and $\varepsilon > 0$, if* 2-MinΔTSP *is $(\alpha, 2-\varepsilon)$-approximable, then* MinΔTSPPst *is α-approximable.*

Proof Suppose there exists an $(\alpha, 2-\varepsilon)$-approximation algorithm for 2-MinΔTSP. Let $G = (V, E, w)$ be an $\mathbb{N}$-labeled complete undirected graph with metric w and let P be a Hamiltonian path in G that connects the vertices s and t. We show how to obtain an α-approximation of P by the following steps.

Step 1: Construction of a reduction graph H. We construct an exact copy $G' = (V', E', w')$ of G, where we denote the copy of $v \in V$ by v'. With $r = 1 + \lceil 1/\varepsilon \rceil > 1/\varepsilon$, we further introduce $2r - 2$ new vertices $S = \{s_1, s_2, \ldots, s_{r-1}\}$ and $T = \{t_1, t_2, \ldots, t_{r-1}\}$. Furthermore, let $s_0 = s$, $s_r = s'$, $t_0 = t$, $t_r = t'$, and define $E_S = \{\{s_0, s_1\}, \{s_1, s_2\}, \ldots, \{s_{r-1}, s_r\}\}$ and $E_T = \{\{t_0, t_1\}, \{t_1, t_2\}, \ldots, \{t_{r-1}, t_r\}\}$. Our reduction graph H will have the vertex set $V^H = V \cup V' \cup S \cup T$ and the edge set $E^H = \{\{u, v\} \mid u, v \in V^H \wedge u \neq v\}$. We will make sure that we find a Hamiltonian cycle that uses the edges E_S and E_T and hence enters and exits G and G' at the vertices s, s', t and t'. Figure 4.2 shows the structure of the graph H.

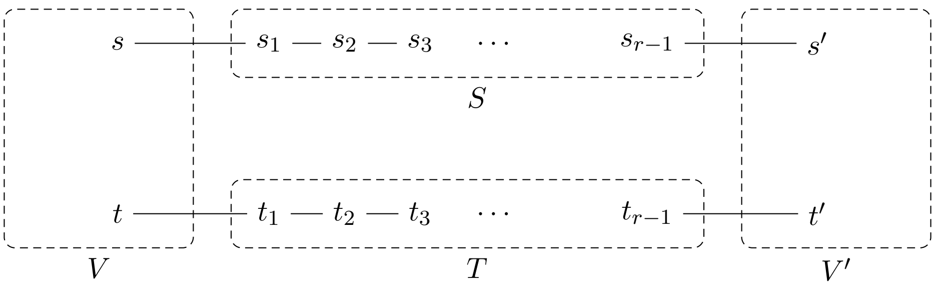

s
s_1 — s_2 — s_3 ⋯ s_{r-1}
s'
S
t
t_1 — t_2 — t_3 ⋯ t_{r-1}
t'
V
T
V'

Figure 4.2: Structure of the graph H. We obtain a copy $G' = (V', E')$ of the graph $G = (V, E)$ and then connect the vertices s with s' and t with t' by new paths that consist of $2(r-1)$ new vertices $S = \{s_1, \ldots, s_{r-1}\}$ and $T = \{t_1, \ldots, t_{r-1}\}$ and $2r$ new edges E_S and E_T. The sets V, V', S, T partition the graph H.

In the remainder of this step, we define a weight function $w^H \colon E^H \to \mathbb{N}^2$ that is metric in both components.

Let $\tilde{E} = E \cup E' \cup E_S \cup E_T$ and define $\tilde{w} \colon \tilde{E} \to \mathbb{N}^2$ by

$$\tilde{w}(e) = \begin{cases} (w(e), 0) & \text{if } e \in E, \\ (w'(e), 0) & \text{if } e \in E', \text{ and} \\ (0, 1) & \text{if } e \in E_S \cup E_T. \end{cases}$$

We extend $\tilde{w}$ to sets $F \subseteq \tilde{E}$ by $\tilde{w}(F) = \sum_{e \in F} \tilde{w}(e)$.

Claim 4.5.2 *For all $u, v \in V^H$ with $u \neq v$ and $e = \{u, v\}$ there exists a path $P_e \subseteq \tilde{E}$ connecting u and v such that $\tilde{w}(P_e) \leq \tilde{w}(P')$ for every path $P' \subseteq \tilde{E}$ connecting u and v.*

Proof Let $u, v \in V^H$ such that $u \neq v$ and let $e = \{u, v\}$. First, observe that there exists a path in $\tilde{E}$ connecting u and v. It remains to show that a minimal path exists. For this purpose, recall that w is metric. We have the following cases.

- $u, v \in V$. For $P_e = \{\{u, v\}\}$ it holds that $\tilde{w}_2(P_e) = 0$. We also show minimality in $\tilde{w}_1$. Suppose that $P' \subseteq \tilde{E}$ connects u and v. If $P' \subseteq E$ then we have $\tilde{w}_1(P') = w(P') \geq w(\{u, v\}) = \tilde{w}_1(\{u, v\}) = \tilde{w}_1(P_e)$. If $P' \not\subseteq E$, then P' covers s, s', t', and t. Hence P' contains paths P_1, P_2, P_3 such that P_1 and P_3 connect u with s (or t) and v with t (or s), and P_2 connects s' with t'. Then we have $\tilde{w}_1(P') = \tilde{w}_1(P_1) + \tilde{w}_1(P_2) + \tilde{w}_1(P_3) \geq w(P_1) + w(\{s, t\}) + w(P_3) \geq w(\{u, v\}) = \tilde{w}_1(P_e)$. Hence in both cases it holds that $\tilde{w}(P_e) \leq \tilde{w}(P')$.
- $u, v \in V'$. Analogously to the case where $u, v \in V$.
- $u, v \in S \cup \{s, s'\}$. Let $P_e \subseteq E_S$ be the unique path connecting u and v, hence $\tilde{w}(P_e) \leq (0, r)$. For every path $P' \subseteq \tilde{E}$ connecting u and v it holds that if $P' \neq P_e$ then $E_T \subseteq P'$, hence $\tilde{w}_2(P') \geq r$, and thus $\tilde{w}(P_e) \leq \tilde{w}(P')$.
- $u, v \in T \cup \{t, t'\}$. Analogously to the case where $u, v \in S \cup \{s, s'\}$.
- $u \in V - \{s\}$ and $v \in S \cup \{s'\}$. Let $P_e = \{u, s\} \cup P_{\{s,v\}}$. If $P' \subseteq \tilde{E}$ is a path connecting u and v with $E_T \subseteq P'$, then $\tilde{w}(P_e) = \tilde{w}(\{u, s\}) + \tilde{w}(P_{\{s,s'\}}) \leq \tilde{w}(\{u, t\}) + \tilde{w}(\{t, s\}) + (0, r) = \tilde{w}(\{u, t\}) + \tilde{w}(\{t', s'\}) + \tilde{w}(P_{\{t,t'\}}) \leq \tilde{w}(P')$. On the other hand, if $P' \subseteq \tilde{E}$ is a path connecting u and v with $E_T \not\subseteq P'$, then P' consists of a first part connecting u and s and a second part connecting s and v, and from the previous cases it follows that $\tilde{w}(P_e) \leq \tilde{w}(P')$.
- $u \in V - \{t\}$ and $v \in T \cup \{t'\}$, and all symmetric cases where u and v are switched or where we consider V'. Analogously to the case where $u \in V - \{s\}$ and $v \in S \cup \{s'\}$.
- $u \in V$ and $v \in V'$, or $u \in V'$ and $v \in V$. Every path $P' \subseteq \tilde{E}$ connecting u and v must either contain E_T or E_S, hence $\tilde{w}_2(P') = r$. Choose P_e as such a path P' with $\tilde{w}_1(P')$ minimal.
- $u \in S$ and $v \in T$, or $u \in T$ and $v \in S$. For every path $P' \subseteq \tilde{E}$ connecting u and v there exists a path $P'' \subseteq \tilde{E}$ connecting u and v with $\tilde{w}_2(P'') = \tilde{w}_2(P')$ and $\tilde{w}_1(P'') = \tilde{w}_1(\{s, t\}) \leq \tilde{w}_1(P')$. Choose P_e as such a path P'' with $\tilde{w}_1(P'') = \tilde{w}_1(\{s, t\})$ and $\tilde{w}_2(P'')$ minimal.

□

With Claim 4.5.2 we can now define the weight function $w^H \colon E^H \to \mathbb{N}^2$ by $w^H(e) = \tilde{w}(P_e)$ for all $e \in E^H$, where P_e is a minimal path in $\tilde{E}$ guaranteed to exist by Claim 4.5.2. This in particular means that w^H is metric in both objectives. We define the $\mathbb{N}^2$-labeled complete reduction graph by $H = (V^H, E^H, w^H)$.

Step 2: Construction of a Hamiltonian cycle C in H. Since w^H is metric in each component, we can call the $(\alpha, 2-\varepsilon)$-approximation algorithm for 2-MINΔTSP on input H. Recall that P is a Hamiltonian path in G and let P' denote the copy of P in G'. Observe that $P \cup P' \cup E_S \cup E_T$ is a Hamiltonian cycle in H with weight $(2w(P), 2r)$. Hence the $(\alpha, 2-\varepsilon)$-approximation algorithm for 2-MINΔTSP must return an $(\alpha, 2-\varepsilon)$-approximation of it. So we obtain a Hamiltonian cycle C of H such that $w_1^H(C) \leq 2\alpha w(P)$ and $w_2^H(C) \leq 4r - 2\varepsilon r$.

Step 3: Transformation of C into a closed spanning walk W using only edges from $\tilde{E}$. Recall that $\{V, V', S, T\}$ is a partition of V^H. Consider an arbitrary edge $e = \{u, v\} \in C$ such that $e \notin \tilde{E}$, and recall that $w^H(e) = \tilde{w}(P_e) = w^H(P_e)$, where $P_e \subseteq \tilde{E}$ is the path connecting u and v from Claim 4.5.2. We replace each edge $e \in C - \tilde{E}$ with the according path P_e and obtain a closed spanning walk W with $w^H(W) = w^H(C)$ that uses only edges from $\tilde{E}$.

Step 4: Extraction of a Hamiltonian path P' from s to t. For $e \in E^H$, let $u(e)$ denote the number of times e is used in W, and for $v \in V^H$, let $d(v)$ denote the degree of v in W (where W is considered as a multigraph). Furthermore, $d(V) = u(\{s, s_1\}) + u(\{t, t_1\})$ and $d(V') = u(\{s_{r-1}, s'\}) + u(\{t_{r-1}, t'\})$. We have the following claims.

Claim 4.5.3 *The degrees $d(v)$ for every vertex v and $d(V)$ and $d(V')$ are all even.*

Proof This holds because W is a closed spanning walk. □

Claim 4.5.4 *The parity of $u(e)$ is the same for all $e \in E_S \cup E_T$.*

Proof We first show that the parity of $u(e)$ is the same for all edges $e \in E_S$. So assume this is not the case. Then there exists a vertex s_i such that $u(\{s_{i-1}, s_i\})$ is odd if and only if $u(\{s_i, s_{i+1}\})$ is even. Hence $d(s_i)$ is odd, which contradicts Claim 4.5.3. Analogously we show that the parity of $u(e)$ is the same for all edges $e \in E_T$. Now assume that the parity of $u(\{s, s_1\})$ is different from the parity of $u(\{t, t_1\})$. Then, $d(V)$ is odd, which contradicts Claim 4.5.3. Since the parity of $u(e)$ is equal for all $e \in E_S$, the parity of $u(e)$ is equal for all $e \in E_T$, and the parity of $u(\{s, s_1\})$ is equal to the parity of $u(\{t, t_1\})$, the claim follows. □

Claim 4.5.5 *There can be at most one $e \in E_S \cup E_T$ such that $u(e) = 0$.*

Proof If there were two such edges, W would not be connected. □

Claim 4.5.6 *All $e \in E_S \cup E_T$ have odd usage count $u(e)$.*

Proof Suppose this is not the case, hence all $e \in E_S \cup E_T$ have even usage count $u(e)$. This means that $u(e) \geq 2$ for all edges with at most one exception (Claim 4.5.5) and thus

$$w_2^H(C) = w_2^H(W) = \sum_{e \in E_S \cup E_T} u(e) \geq (2r-1) \cdot 2 = 4r - 2 > 4r - 2\varepsilon r \qquad (\text{since } r > 1/\varepsilon)$$

which contradicts the approximation rate of C guaranteed in Step 2. □

We can now extract a Hamiltonian path in G as follows. Recall that $w_1^H(W) = w_1^H(C) \leq 2\alpha w(P)$. We interpret W as the multigraph M. We remove the edges $E_S \cup E_T$ and the vertices $S \cup T$ from M to obtain two connected components, where we interpret the component with vertices V as the multigraph M_V and the component with vertices V' as the multigraph $M_{V'}$. Since all edges in $E_S \cup E_T$ had odd usage count, s and t are the only vertices in M_V with odd degree, and s' and t' are the only vertices in $M_{V'}$ with odd degree. Hence we can find a spanning walk W_1 in M_V with start and end vertices s and t, and analogously a spanning walk W_2 in $M_{V'}$ with start and end vertices s' and t'. We transform W_1 and W_2 into Hamiltonian paths P_1 and P_2 by taking shortcuts. Note that in each step we did not increase weight, hence $w_1^H(P_1)+w_1^H(P_2) \leq w_1^H(W_1)+w_1^H(W_2) \leq w_1^H(W) = w_1^H(C) \leq 2\alpha w(P)$, hence there exists some $i \in \{1,2\}$ with $w_1^H(P_i) \leq \alpha w(P)$. Since $w_1^H(P_1) = w(P_1)$ and P_2 is the copy of some $\tilde{P}_2 \subseteq E$ with $w_1^H(P_2) = w_1^H(\tilde{P}_2) = w(\tilde{P}_2)$, we have found a path Hamiltonian path in G connecting s and t with weight at most $\alpha w(P)$. □

Corollary 4.5.7 *For every $\varepsilon_1, \varepsilon_2 > 0$, if* Multigraph 2-MinTSP *is* $(\frac{(1+\sqrt{5})}{2} - \varepsilon_1, 2 - \varepsilon_2)$-*approximable, then* MinΔTSPPst *is* $(\frac{(1+\sqrt{5})}{2} - \varepsilon_1)$-*approximable.*

4.5.2 Lower Bound Arguments for TSPP

Analogously to Theorem 4.5.1 we show that MinΔTSPPs and MinΔTSPPst can be reduced to 2-MinΔTSPP and 2-MinΔTSPPs by an approximation preserving reduction. By a second weight function we make sure that the end vertex constraints are satisfied. We obtain the following result.

Theorem 4.5.8 *Let $\alpha > 1$ and $\varepsilon > 0$.*

1. *If* 2-MinΔTSPP *is* $(\alpha, 3/2 - \varepsilon)$-*approximable, then* MinΔTSPPst *is* α-*approximable.*
2. *If* 2-MinΔTSPP *is* $(\alpha, 2 - \varepsilon)$-*approximable, then* MinΔTSPPs *is* α-*approximable.*
3. *If* 2-MinΔTSPPs *is* $(\alpha, 2 - \varepsilon)$-*approximable, then* MinΔTSPPst *is* α-*approximable.*

Proof The proofs are similar to the proof of Theorem 4.5.1. We will give a full proof of the first item and show the differences for the remaining items.

1. We reduce MinΔTSPPst to 2-MinΔTSPP. So suppose there exists an $(\alpha, 3/2 - \varepsilon)$-approximation algorithm for 2-MinΔTSPP. Let $G = (V, E, w)$ be an $\mathbb{N}$-labeled complete undirected graph with metric w and let P be a Hamiltonian path in G that connects vertices s and t. We show how to obtain an α-approximation of P that connects vertices s and t.
   We define a new weight function $w' \colon E \to \mathbb{N}^2$ as follows. For each $e \in E$, define $w_1'(e) = w(e)$ and
   $$w_2'(\{u,v\}) = \begin{cases} 0 & \text{if } \{u,v\} \cap \{s,t\} = \emptyset, \\ 1 & \text{if } |\{u,v\} \cap \{s,t\}| = 1, \text{ and} \\ 2 & \text{if } \{u,v\} = \{s,t\}. \end{cases}$$
   Both w_1' and w_2' are metric functions on V. Moreover, for every Hamiltonian path P' connecting s and t it holds that $w_2'(P') = 2$, which includes P. On the other hand, all remaining Hamiltonian paths P'' must have $w_2'(P'') \geq 3$.
   We call the $(\alpha, 3/2 - \varepsilon)$-approximation algorithm for 2-MinΔTSPP on input (V, E, w'). Hence we find a Hamiltonian path P' that approximates P. If P' does not connect s and t, we have $w_2'(P') \geq 3$, contradicting $w_2'(P') \leq (3/2 - \varepsilon) w_2'(P) = (3/2 - \varepsilon)2 < 3$. Hence P' connects s and t. Moreover we have $w(P') = w_1'(P') \leq \alpha w_1'(P) = \alpha w(P)$.

2. Analogously to the first item, we reduce MINΔTSPPS to 2-MINΔTSPP. We define $w''\colon E \to \mathbb{N}^2$ such that for all $e \in E$, $w''_1(e) = w(e)$ and
$$w''_2(\{u,v\}) = \begin{cases} 0 & \text{if } s \notin \{u,v\}, \text{ and} \\ 1 & \text{otherwise.} \end{cases}$$
Again, w''_1 and w''_2 are metric. Moreover, for every Hamiltonian path P it holds that $w''_2(P) = 1$ if P starts or ends at s, and $w''_2(P) = 2$ otherwise. Hence, for every Hamiltonian path P starting at s it holds that $w''_2(P) = 1$, and the $(\alpha, 2-\varepsilon)$-approximation algorithm for 2-MINΔTSPP on input (V, E, w'') finds a Hamiltonian path such that $w(P') = w''_1(P') \leq \alpha w''_1(P) = \alpha w(P)$ and $w''_2(P') = 1$, hence P' also starts at s.
3. Analogously to the second item, we reduce MINΔTSPPST to 2-MINΔTSPPS. We use w'' for the reduction and call the $(\alpha, 2-\varepsilon)$-approximation algorithm for 2-MINΔTSPPS on input (V, E, w'') and t. Hence, every output path starts or ends at t. By the argumentation from the second item, for every Hamiltonian path P starting at s, we obtain a Hamiltonian path P' that starts at s and has weight $w(P') \leq \alpha w(P)$. Since every output path starts or ends at t, the result follows.

□

Corollary 4.5.9 *Let* $\varepsilon_1, \varepsilon_2 > 0$.

1. *If* MULTIGRAPH 2-MINTSPP *is* $(\frac{1+\sqrt{5}}{2} - \varepsilon_1, 3/2 - \varepsilon_2)$*-approximable, then* MINΔTSPPST *is* $(\frac{1+\sqrt{5}}{2} - \varepsilon_1)$*-approximable.*
2. *If* MULTIGRAPH 2-MINTSPP *is* $(3/2 - \varepsilon_1, 2 - \varepsilon_2)$*-approximable, then* MINΔTSPPS *is* $(3/2 - \varepsilon_1)$*-approximable.*
3. *If* MULTIGRAPH 2-MINTSPPS *is* $(\frac{1+\sqrt{5}}{2} - \varepsilon_1, 2 - \varepsilon_2)$*-approximable, then* MINΔTSPPST *is* $(\frac{1+\sqrt{5}}{2} - \varepsilon_1)$*-approximable.*

4.6 Summary and Discussion

We introduced a multigraph model for multiobjective minimum traveling salesman problems that captures realistic scenarios and covers the conventional definition that can be found in the literature [MR09, Ehr00]. For the two-objective case, we showed new randomized and deterministic algorithms and hence improved the previously best known upper bound on the approximation ratio for these problems. As a byproduct, we demonstrated how to generalize known approximation schemes for multiobjective matching, spanning tree, and shortest path problems to the multigraphs variants of these problems.

We further showed reductions from well-studied single-objective problems to the two-objective traveling salesman problems considered here. These reductions generally show that significant improvements of our results are difficult to obtain, as they would lead to new results on well-studied, single-objective problems such as MINΔTSPPS or MINΔTSPPST.

In the case of MULTIGRAPH 2-MINTSP, our results are particularly interesting. On the one hand, we have a randomized $(3/2 + \varepsilon, 2)$-approximation algorithm and a deterministic $(2, 2)$-approximation algorithm for this problem, while on the other hand, we know that an $(1/2 + \sqrt{5}/2 - \varepsilon_1, 2 - \varepsilon_2)$-approximation algorithm would improve the currently best known approximation algorithm for MINΔTSPPST. We have no argument for or against the existence of an α-approximation algorithm for MULTIGRAPH 2-MINTSP, where $1,618 \approx 1/2 + \sqrt{5}/2 \leq \alpha < 2$. The search for such an algorithm remains a challenging task.

Chapter 5

Applications of Discrepancy Theory

Introduction to Maximum Traveling Salesman Problems We now shift our focus to the approximability of multiobjective maximization variants of the traveling salesman problem. We distinguish between the symmetric and the asymmetric variant. In the single-objective setting, we denote these problems by MaxSTSP and MaxATSP, respectively. The input for MaxSTSP is a complete, $\mathbb{N}$-labeled, undirected graph, and the input for MaxATSP is a complete, $\mathbb{N}$-labeled, directed graph. In both cases we want to obtain a Hamiltonian cycle of maximum weight. The maximization problems admit approximations already in this very general setting, where in particular no metric weight assumptions are required.

A common strategy to approximate maximum traveling salesman problems works on cycle covers. Note that every Hamiltonian cycle is a cycle cover that consists of a single cycle. In contrast to Hamiltonian cycles of maximum weight, a maximum weight cycle cover for a directed or undirected graph can be computed in polynomial time (see the work of Manthey [Man05] for a detailed discussion of the cycle cover problem and its variants). A general approach to approximate a maximum weight Hamiltonian cycle computes a cycle cover of maximum weight, then breaks the cycles into paths by removing an edge per cycle, and connects the remaining paths to a single Hamiltonian cycle. In 1979, Fisher, Nemhauser and Wolsey [FNW79] argued that removing the lightest edge in each cycle results in a $1/2$-approximation algorithm for MaxATSP. Note that each cycle in a directed graph consists of at least two edges, hence by removing the lightest edge per cycle we lose at most half of the weight of the maximum weight cycle cover, which in turn must be at least as heavy as every maximum weight Hamiltonian cycle. This shows the approximation ratio of $1/2$. Since undirected cycles always contain at least three edges, the same approach applies to MaxSTSP, which hence is $2/3$-approximable. In Figure 5.1 we give an example application of such an algorithm.

Since then, many improvements were achieved. In 2005, Kaplan et al. [KLSS05] showed a $2/3$-approximation for MaxATSP, which was simplified by Paluch et al. [PEvZ12] and in 2014 improved by Paluch [Pal14], who gave a $3/4$-approximation algorithm for MaxATSP. The currently best known approximation ratio of $7/9$ for MaxSTSP is due to Paluch, Mucha and Madry [PMM09].

Maximum Traveling Salesman Problems with Multiple Objectives We obtain the multiobjective versions k-MaxATSP and k-MaxSTSP by considering complete, $\mathbb{N}^k$-labeled, directed and undirected input graphs, respectively, where $k \geq 1$. Note that the approximation scheme for maximum weight perfect matching from Theorem 3.2.1 is randomized. As a consequence, most cycle cover based algorithms for k-MaxATSP and k-MaxSTSP that compute cycle covers by a reduction to the randomized matching algorithms are also randomized. More-

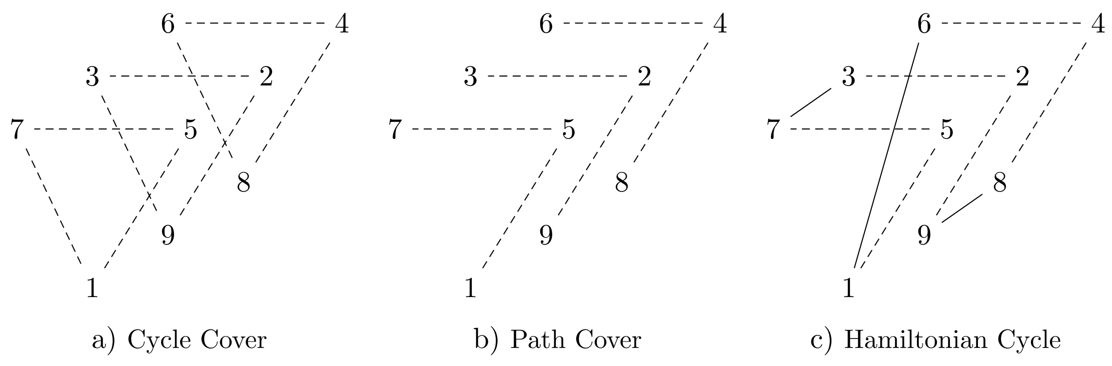


Figure 5.1: Example for the execution of a cycle cover based algorithm. Suppose we have computed a cycle cover of an undirected, complete graph as a first step. Every cycle contains at least three edges. In the next step we remove the shortest edge per cycle. Since every cycle consisted of at least three edges and since we removed the shortest edge, we removed at most $1/3$ of the overall distance. We obtain a set of paths. In the last step we arbitrarily connect the paths to a Hamiltonian cycle.

over, a trivial adaption of the above cycle cover based approximation algorithms for MaxATSP and MaxSTSP to the case of multiple objectives fails, because the term "lightest edge" is not well-defined in the multiobjective setting.

Bläser et al. [BMP08] were the first to give a randomized $(1-\varepsilon)/k$-approximation for k-MaxSTSP and a randomized $(1-\varepsilon)/(k+1)$-approximation for k-MaxATSP, where $k \geq 2$ and $\varepsilon > 0$. Manthey [Man12b] significantly improved these results by showing a randomized $(2-\varepsilon)/3$-approximation for k-MaxSTSP and a randomized $(1-\varepsilon)/2$-approximation for k-MaxATSP, where the approximation ratio is independent on the number of objectives.

Note that all above mentioned approximation algorithms for k-MaxATSP and k-MaxSTSP are randomized. Manthey [Man12a] further studied deterministic approximation algorithms for both problems. For multiobjective matching problems and hence also for multiobjective cycle cover problems, no FPTAS is known so far, so deterministically computing approximations for k-MaxATSP and k-MaxSTSP becomes significantly harder. Manthey [Man12a] showed that k-MaxATSP is deterministically $(1-\varepsilon)/(4k-2)$-approximable, and k-MaxSTSP is deterministically $(1-\varepsilon)/2k$-approximable, where $k \geq 2$ and $\varepsilon > 0$. Moreover, in the two-objective case, he obtained a deterministic $(1-\varepsilon)/4$-approximation algorithm for 2-MaxATSP and a deterministic $(3-\varepsilon)/8$-approximation algorithm for 2-MaxSTSP. We will study deterministic algorithms for k-MaxATSP and k-MaxSTSP in the next chapter.

Discrepancy Theory in Multiobjective Optimization A trivial adaption of the cycle cover based algorithms to multiple objectives fails, because in this setting we cannot choose the lightest edge per cycle. In particular, it might be the case that the weights are distributed in such a way that for every edge, whenever the weight in one objective is low, then it is high in another objective. So for cycle covers with multiobjective edge weights, the following questions arise.

1. Is it possible to choose a set of edges with one edge per cycle such that the overall weight is at most half of the weight of the entire cycle cover *in each objective*?
2. If this is possible, can we compute such a set of edges in polynomial time?

We show that with a slight deviation, both questions can be answered positively. Moreover, our results are not restricted to the graph-theoretic setting, but rather generally apply to multidimensional vectors. Our proof is based on discrepancy theory. Here, a well-known result by Beck and Fiala [BF81] considers the deviation of discrete colorings from an "ideal coloring" in a set system. Suppose we are given a universe $U = \{u_1, \dots, u_n\}$ and a set system $S = \{S_1, \dots, S_m\} \subseteq 2^U$. Beck and Fiala show that whenever each element of U is contained in no more than t sets of S, then there exists a coloring $\chi\colon U \to \{+1, -1\}$ such that for each $S_j \in S$ it holds that $\text{abs}(\sum_{u \in S_j} \chi(u)) \leq 2t - 1$. Hence in such a setting, the discrepancy between the ideal situation where each set in S has a balanced coloring and the best existing coloring is at most $2t - 1$. Doerr and Srivastav [DS03] further extend the result of Beck and Fiala to a multicolor setting. We will include their proof and show how to obtain our result as a corollary.

Contributions We summarize the contributions of this chapter as follows.

1. *A General Tool for Balancing Vectors.* The problem of having incomparable weight vectors appears frequently in multiobjective optimization. Using the discrepancy-theoretic results by Doerr and Srivastav [DS03] we provide a tool for choosing a subset of given vectors such that the subset is "balanced" in each component. This tool enables us to translate some single-objective approximation algorithms to the multiobjective case. Although we developed our balancing tool to obtain approximations for the maximum traveling salesman problems with multiple objectives, it also applies to further optimization problems. In addition to the traveling salesman problems, we will show its application to a multiobjective version of MAXSAT.
2. *Improved Approximations.* We obtain cycle cover based algorithms for k-MAXATSP and k-MAXSTSP that generalize the basic approximations of Fischer, Nemhauser and Wolsey [FNW79] to the k-objective setting, where $k \geq 1$. Since there are FPRAS that approximate the cycle cover problems, we can even fix heavy edges before executing our algorithm to remove the small approximation error introduced by the cycle cover algorithm. We obtain a randomized $1/2$-approximation for k-MAXATSP and a randomized $2/3$-approximation for k-MAXSTSP, which improves the results of Manthey [Man12b]. Moreover, we apply our vector balancing result in a similar way to a k-objective version of MAXSAT, which we denote by k-MAXSAT. For k-MAXSAT, we obtain a deterministic $1/2$-approximation. We summarize our approximation results in Table 5.1.

Problem	Deterministic Approximation	Randomized Approximation	Reference
k-MAXATSP		$1/2$	5.4.2
k-MAXSTSP		$2/3$	5.4.2
k-MAXSAT	$1/2$		5.5.2

Table 5.1: Approximation ratios shown in this chapter, where $k \geq 1$.

Organization of this Chapter In Section 5.1 we give a precise definition of the problems we consider in this chapter. Recall that we consider cycle cover based algorithms. In Section 5.2 we will show how to obtain randomized approximation schemes for the cycle cover problems we use in our approximation algorithms. In Section 5.3 we show how to obtain the vector balancing results. For completeness we include a proof of the result by Doerr and Srivastav [DS03], which also shows how to compute a vector balancing in polynomial time. In Section 5.4 we will then show our randomized approximation results for k-MAXATSP and k-MAXSTSP, and in Section 5.5 we will show how to obtain a deterministic approximation for k-MAXSAT using the same balancing result and a similar algorithmic approach. We summarize and discuss our results in Section 5.6.

5.1 Problem Definitions

Let $k \geq 1$ denote the number of objectives. The k-objective traveling salesman problems k-MaxATSP and k-MaxSTSP are defined as follows.

k-Objective Maximum Asymmetric Traveling Salesman Problem

Notation:	k-MaxATSP
Instance:	$\mathbb{N}^k$-labeled directed complete graph $G = (V, E, w)$
Solution:	Hamiltonian cycle $C \subseteq E$ of G
Objective:	maximize $w(C)$

k-Objective Maximum Symmetric Traveling Salesman Problem

Notation:	k-MaxSTSP
Instance:	$\mathbb{N}^k$-labeled undirected complete graph $G = (V, E, w)$
Solution:	Hamiltonian cycle $C \subseteq E$ of G
Objective:	maximize $w(C)$

While this chapter mainly focuses on approximation algorithms for k-MaxATSP and k-MaxSTSP, the balancing tool we develop is generally applicable. We will show a similar approximation of the k-objective maximum weight satisfiability problem. Recall the definition of variables, literals, clauses, formulas, and truth assignments from Section 2.3. The k-objective maximum weight satisfiability problem is defined as follows.

k-Objective Maximum Weighted Satisfiability

Notation:	k-MaxSAT
Instance:	formula H in CNF over a set of variables V and weight function $w \colon H \to \mathbb{N}^k$
Solution:	truth assignment $I \colon V \to \{0, 1\}$
Objective:	maximize $w(I) = \sum_{C \in H \colon I(C)=1} w(C)$

Our approximation algorithms for k-MaxATSP and k-MaxSTSP are based on multiobjective cycle cover approximations. We consider the following cycle cover problems, where $c \geq 0$.

k-Objective Maximum Weight Undirected Edge-Fixed c-Cycle Cover

Notation:	k-MaxUFCC$_c$
Instance:	$\mathbb{N}^k$-labeled undirected graph $G = (V, E, w)$ and $F \subseteq E$
Solution:	cycle cover C of G with $F \subseteq C$ and at least c edges per cycle
Objective:	maximize $w(C)$

k-Objective Maximum Weight Directed Edge-Fixed c-Cycle Cover

Notation:	k-MaxDFCC$_c$
Instance:	$\mathbb{N}^k$-labeled directed graph $G = (V, E, w)$ and $F \subseteq E$
Solution:	cycle cover C of G with $F \subseteq C$ and at least c edges per cycle
Objective:	maximize $w(C)$

If we restrict the above cycle cover problems to instances with $F = \emptyset$, we obtain the *k-objective maximum weight undirected c-cycle cover problem (k-MaxUCC$_c$)* and the *k-objective maximum weight directed c-cycle cover problem (k-MaxDCC$_c$)*. In directed graphs, every cycle consists of at least 2 edges, and in undirected graphs, every cycle consists of at least 3 edges. Hence, for small c we can omit the index and define k-MaxDCC as k-MaxDCC$_2$ and k-MaxUCC as k-MaxUCC$_3$. Analogously we define k-MaxDFCC and k-MaxUFCC as k-MaxDFCC$_2$ and k-MaxUFCC$_3$, respectively.

These cycle cover problems will be reduced to the multiobjective maximum perfect matching problem, which we define as follows.

k-Objective Maximum Weight Perfect Matching

Notation:	k-MaxPM
Instance:	$\mathbb{N}^k$-labeled undirected graph $G = (V, E, w)$
Solution:	perfect matching $M \subseteq E$ of G
Objective:	maximize $w(M)$

5.2 Approximation of Cycle Cover Problems

Recall that by Papadimitriou and Yannakakis [PY00], for every $k \geq 1$ there exists an FPRAS for k-MaxPM (cf. Theorem 3.2.1). In this section we show that the matching FPRAS can be used to obtain FPRAS for the cycle cover problems we defined in the previous section.

We first consider the cycle cover problems without fixed edge sets and without minimum cycle length restrictions. We show reductions to the perfect matching problem that are analogous to the single-objective case. First consider the case of directed graphs. Every directed loop-free graph can be transformed into an undirected bipartite graph, by splitting each vertex v into two new vertices v_1 and v_2 and connecting all incoming edges of v to v_1 and all outgoing edges of v to v_2. Then, every perfect matching of the undirected bipartite graph corresponds to a cycle cover of the directed loop-free graph. Now, consider the case of undirected graphs. Here, we can simply use Tutte's reduction [Tut54] that reduces the cycle cover problem on undirected graphs to the perfect matching problem on undirected graphs. Below we show how to adapt these reductions to the case of $\mathbb{N}^k$-labeled graphs. Together with Theorem 3.2.1 we obtain randomized approximation schemes for k-MaxDCC and k-MaxUCC.

Lemma 5.2.1 *For every $k \geq 1$ there exists an FPRAS for* k-MaxDCC *and* k-MaxUCC.

Proof We first describe the FPRAS for k-MaxDCC. On input of an $\mathbb{N}^k$-labeled, directed graph $G = (V, E, w)$ and $\varepsilon > 0$, we define the $\mathbb{N}^k$-labeled undirected graph $G' = (V', E', w')$ by

$$\begin{aligned} V' &= \{1,2\} \times V \\ E' &= \{\{(1,u),(2,v)\} \mid (u,v) \in E \text{ and } u \neq v\} \\ w'(e) &= w(u,v) \qquad \text{for each } e = \{(1,u),(2,v)\},\ (u,v) \in E. \end{aligned}$$

We call the FPRAS for k-MaxPM from Theorem 3.2.1 on G' and ε. Suppose the FPRAS succeeds, then we obtain an ε-approximate set of perfect matchings of G'. For each approximate perfect matching M' we output $\{(u,v) \mid \{(1,u),(2,v)\} \in M'\}$.

Let C be a cycle cover of G. Then, $M = \{\{(1,u),(2,v)\} \mid (u,v) \in C\}$ is a perfect matching of G' with $w'(M) = w(C)$. Hence we find a perfect matching M' of G' with $w'(M') \geq (1-\varepsilon)w'(M)$. Then, $C' = \{(u,v) \mid \{(1,u),(2,v)\} \in M'\}$ is a cycle cover of G with $w(C') = w'(M') \geq (1-\varepsilon)w'(M) = (1-\varepsilon)w(C)$.

Note that the FPRAS from Theorem 3.2.1 succeeds with probability at least $1/2$, hence our algorithm is an FPRAS for k-MaxDCC.

Next we describe the FPRAS for k-MaxUCC. Here, on input of an $\mathbb{N}^k$-labeled, undirected graph $G = (V, E, w)$ and $\varepsilon > 0$, we construct an undirected graph $G' = (V', E', w')$ as defined by Tutte's reduction [Tut54]. For each cycle cover C in G there exists a perfect matching M in G' with $w(C) = w'(M')$ and vice versa. Hence we can again apply the FPRAS from Theorem 3.2.1 to the reduction graph and ε. This shows that there also exists an FPRAS for k-MaxUCC. □

We obtain FPRAS for the problems k-MaxDFCC and k-MaxUFCC by a simple reduction to k-MaxDCC and k-MaxUCC, respectively. We introduce an additional weight function and set the additional weight to 1 for the fixed edges, and to 0 otherwise. By choosing the approximation parameter $\varepsilon > 0$ sufficiently small, we make sure that all edges with weight 1 have to appear in each solution. We hence have the following result.

Lemma 5.2.2 *For every $k \geq 1$ there exists an FPRAS for k-MaxDFCC and k-MaxUFCC.*

Proof On input of an $\mathbb{N}^k$-labeled graph $G = (V, E, w)$, $F \subseteq E$, and $\varepsilon > 0$, define $w' \colon E \to \mathbb{N}^{k+1}$,

$$w'_i(e) = \begin{cases} w_i(e) & \text{for } 1 \leq i \leq k \\ 1 & \text{for } i = k+1 \text{ and } e \in F \\ 0 & \text{for } i = k+1 \text{ and } e \notin F. \end{cases}$$

Call the FPRAS from Lemma 5.2.1 on the $(k+1)$-objective cycle cover instance (V, E, w') with approximation parameter $\varepsilon' = \min\{\varepsilon, 1/(|F|+1)\}$, and return all cycle covers C with $F \subseteq C$.

We only return cycle covers C with $F \subseteq C$, so the algorithm is correct. Since the FPRAS from Lemma 5.2.1 succeeds with probability at least $1/2$, our algorithm also succeeds with probability at least $1/2$. It remains to argue for the approximation ratio. So suppose the algorithm succeeds, and consider some cycle cover C of G with $F \subseteq C$. Then we find a cycle cover C' of G with $w'(C') \geq (1 - \varepsilon')w'(C)$. By the definition of w' this means

$$|F| \geq w'_{k+1}(C') \geq (1 - \varepsilon')w'_{k+1}(C) \geq \left(1 - \frac{1}{|F|+1}\right)|F| > |F| - 1,$$

hence we have $w'_{k+1}(C') = |F|$, and this shows $F \subseteq C'$. Moreover, for all $1 \leq i \leq k$ it holds that $w_i(C') = w'_i(C') \geq (1 - \varepsilon')w'_i(C) \geq (1 - \varepsilon)w_i(C)$, so the approximation ratio is correct. This shows that there exists an FPRAS for k-MaxDFCC and k-MaxUFCC. □

5.3 Discrepancy Results

We continue to show how to appropriately combine incomparable weight vectors. This will later help us to select one edge per cycle of a cycle cover that we can remove without losing too much overall weight in any objective. We rely on a result of Doerr and Srivastav [DS03], who provide a polynomial-time algorithm to compute colorings of low discrepancy for matrices of multidimensional vectors. For the sake of completeness, we include their proof below, for which we need to introduce some more terminology.

For a vector $x \in \mathbb{Q}^m$ let $\|x\|_\infty = \max_i(\text{abs}(x_i))$, and for a matrix $A \in \mathbb{Q}^{m \times n}$ let $\|A\|_1 = \max_j \sum_i \text{abs}(a_{ij})$. For $c \geq 2$, $n \geq 1$ let $\overline{M_{c,n}} = \{x \in (\mathbb{Q} \cap [0,1])^{cn} \mid \sum_{k=0}^{c-1} x_{cb-k} = 1 \text{ for all } b \in \{1, \dots, n\}\}$ and $M_{c,n} = \overline{M_{c,n}} \cap \{0,1\}^{cn}$.

Theorem 5.3.1 (Doerr and 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 $\chi = \chi^{(0)} = p \in \overline{M_{c,n}}$ 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 $I := I(\chi) := \{i \in \{1, \dots, m\} \mid \sum_{j \in J(\chi)} \text{abs}(a_{ij}) > 2\Delta\}$. We will ensure that during the rounding process the following conditions are fulfilled (this is clear from the start, because $\chi^{(0)} = p$):

$$(A(p-\chi))_{|I} = 0 \qquad \text{(C1)} \qquad\qquad\qquad \chi \in \overline{M_{c,n}} \qquad \text{(C2)}$$

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 $|B| \le \frac{1}{2}|J|$.

Since

$$|J|\cdot\Delta = \sum_{j\in J}\Delta \ge \sum_{j\in J}\sum_{i=1}^{m}\operatorname{abs}(a_{ij}) \ge \sum_{j\in J}\sum_{i\in I}\operatorname{abs}(a_{ij}) = \sum_{i\in I}\sum_{j\in J}\operatorname{abs}(a_{ij}) > \sum_{i\in I}2\Delta = |I|\cdot 2\Delta$$

it holds that $|I| < \frac{1}{2}|J|$. Consider the inhomogeneous system of linear equations

$$\begin{aligned} &(A(p-\chi))_{|I} = 0 \\ &\sum_{k=0}^{c-1}\chi_{cb-k} = 1 && \text{for } b \in B \end{aligned}$$

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 $\|A(p-\chi)\|_\infty \le 2\Delta$. Let $1 \le i \le m$. Since at the end it holds that $J = \emptyset$, 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 $\operatorname{abs}(\chi_j^{(t)} - \chi_j) < 1$ for all $j \in J(\chi^{(t)})$. Finally note that $\sum_{j\in J(\chi^{(t)})}\operatorname{abs}(a_{ij}) \le 2\Delta$ since $i \notin I(\chi^{(t)})$. Combining these facts, we obtain

$$\begin{aligned} \operatorname{abs}((A(p-\chi))_i) &= \operatorname{abs}((A(p-\chi^{(t)}))_i + (A(\chi^{(t)}-\chi))_i) \\ &= \operatorname{abs}(0 + \sum_{j\in J(\chi^{(t)})} a_{ij}(\chi_j^{(t)} - \chi_j)) \le 2\Delta. \end{aligned}$$

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 the book by Grötschel et al. [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 5.3.2 *There is a polynomial-time algorithm that on input of a set of vectors $v^{j,r} \in \mathbb{Q}^m$ for $1 \le j \le n$, $1 \le r \le c$ computes a coloring $\chi \colon \{1, \dots, n\} \to \{1, \dots, c\}$ such that for each $1 \le i \le m$ it holds that*

$$\mathrm{abs}\left(\frac{1}{c} \sum_{j=1}^{n} \sum_{r=1}^{c} v_i^{j,r} - \sum_{j=1}^{n} v_i^{j,\chi(j)} \right) \le 2m \max_{j,r} \mathrm{abs}(v_i^{j,r}).$$

Proof The result is obvious for $c = 1$. For $c \ge 2$, we use Theorem 5.3.1. Because the error bound is different for each row, we need to scale the rows of the vectors. Let $\delta_i = \max_{j,r} \mathrm{abs}(v_i^{j,r})$ for $1 \le i \le m$. 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 $1 \le i \le cn$. We obtain a coloring $\chi \in \{0,1\}^{cn}$ such that for each $1 \le j \le n$ there is exactly one $1 \le r \le c$ such that $\chi_{c(j-1)+r} = 1$ and it holds that $\|A(p - \chi)\|_\infty \le 2\|A\|_1$. Note that because of the scaling, the largest entry in A is 1 and thus we have $\|A\|_1 \le m$. 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 $1 \le i \le m$:

$$\begin{aligned} 2m\delta_i \ge 2\|\delta_i A\|_1 &\ge \mathrm{abs}((\delta_i A(p - \chi))_i) \\ &= \mathrm{abs}\left(\sum_{j'=1}^{cn} \delta_i a_{ij'} (p_{j'} - \chi_{j'}) \right) = \mathrm{abs}\left(\sum_{j=1}^{n} \sum_{r=1}^{c} \frac{1}{c} v_i^{j,r} - \sum_{j=1}^{n} v_i^{j,\chi'(j)} \right) \end{aligned}$$

□

5.4 Randomized Approximation of k-MaxTSP

With the cycle cover algorithms and the vector balancing results from the last two sections, we are now ready to describe our approximation algorithms for k-MaxATSP and k-MaxSTSP. Generally speaking, our algorithm works in the same way as the basic algorithm by Fisher, Nemhauser and Wolsey [FNW79]. On input of an $\mathbb{N}^k$-labeled complete graph, we first compute a set of approximate cycle covers. For each cycle cover we fix two or three arbitrary edges. Our vector balancing results tell us which edge to remove per cycle such that we do not lose much more than 1/2 (in the case of directed graphs) and 1/3 (in the case of undirected graphs) of the weight of the cycle cover *in each objective*. The remaining paths are then arbitrarily connected to a Hamiltonian cycle.

Given an arbitrary Hamiltonian cycle of the input graph, our cycle cover approximation must contain a $(1 - \varepsilon)$-approximate cycle cover (where $\varepsilon > 0$ is small), and furthermore, we might lose slightly more than 1/2 or 1/3 of the cycle cover weight due to the error bound of the vector balancing result. Hence the above described algorithm computes an approximation with a ratio slightly below 1/2 or 2/3. We remove this error by guessing and fixing a set of heavy edges of the Hamiltonian cycle for each component in advance. We will see that only a constant number of such heavy edges are necessary to balance out the error introduced, hence an exhaustive search is possible in polynomial time. This way we obtain a randomized $1/2$-approximation for k-MaxATSP and a randomized $2/3$-approximation for k-MaxSTSP.

Consider the algorithm k-`MaxTSPApprox`. Our algorithm uses an additional parameter $c \geq 2$ and is generally applicable whenever there exists a cycle cover approximation algorithm that guarantees at least c edges per cycle. We will apply the general algorithm to directed graphs with $c = 2$ and to undirected graphs with $c = 3$.

```
Algorithm: $k$-MaxTSPApprox$(V, E, w)$ with parameter $c \geq 2$

  Input  : $\mathbb{N}^k$-labeled directed/undirected complete graph $G = (V, E, w)$
  Output : Hamiltonian cycles of $G$

1  foreach $F = F_H \cup F_L \subseteq E$ with $3ck \leq |F_H| \leq 3ck^2$, $|F_L| \leq c|F_H|$ do
2      let $\beta \in \mathbb{N}^k$ with $\beta_i = \max\{n \in \mathbb{N} \mid$ there are $3ck$ edges $e \in F_H$ with $w_i(e) \geq n\}$
3      for the current iteration of line 1, let $w' \colon E \to \mathbb{N}^k$ such that
           $w'(e) = \begin{cases} w(e) & \text{if } w(e) \leq \beta \text{ or } e \in F_H\text{, and} \\ 0 & \text{otherwise} \end{cases}$
       for all $e \in E$
4      compute $(1 - 1/|V|)$-approximation $\mathcal{C}$ of $k$-MaxDFCC$_c$ / $k$-MaxUFCC$_c$ on
       $((V, E, w'), F)$
5      foreach cycle cover $C \in \mathcal{C}$ do
6          let $C_1, \ldots, C_r$ denote the cycles in $C$
7          if for each $i \in \{1, \ldots, r\}$, $(C_i - F_H)$ contains a path of length $c$ then
8              foreach $i \in \{1, \ldots, r\}$ do  choose path $e_{i,1}, \ldots, e_{i,c} \in (C_i - F_H)$ arbitrarily
9              compute some coloring $\chi \colon \{1, \ldots, r\} \to \{1, \ldots, c\}$ such that
                   $\sum_{i=1}^{r} w'(e_{i,\chi(i)}) \leq 2k \cdot \beta + \frac{1}{c} \sum_{i=1}^{r} \sum_{j=1}^{c} w'(e_{i,j})$
               and remove the edges $\{e_{i,\chi(i)} \mid 1 \leq i \leq r\}$ from $C$
10             output the remaining edges, connected to a Hamiltonian cycle
```

Lemma 5.4.1 *Let $c \geq 2$ and $k \geq 1$. If there exists an FPRAS for k-*MaxDFCC$_c$ *(resp., k-*MaxUFCC$_c$*), then the algorithm k-**MaxTSPApprox** computes a randomized* $(1 - 1/c)$-*approximation for k-*MaxATSP *(resp., k-*MaxSTSP*) in polynomial time.*

Proof Let $k \geq 1$ and $c \geq 2$. We argue that the algorithm k-`MaxTSPApprox` with parameter c, using a polynomially amplified version of the FPRAS for k-MaxDFCC$_c$ (resp., k-MaxUFCC$_c$) as a subroutine, is a randomized approximation algorithm for k-MaxATSP (resp., k-MaxSTSP) with approximation ratio $(1 - 1/c)$. Our argumentation works for both directed and undirected input graphs. So let $G = (V, E, w)$ be some $\mathbb{N}^k$-labeled input graph with $m = |V|$ sufficiently large.

Observe that for every input we return a set of Hamiltonian cycles, hence the algorithm is correct. We argue for polynomial runtime, success probability, and approximation ratio.

Runtime: 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 polynomially amplified version of the FPRAS on (V, E, w') and F terminates in time polynomial in the length of (V, E, w') and F, which means that the set $\mathcal{C}$ contains only polynomially many cycle covers. Hence, for each iteration of the loop in line 1, the loop in line 5 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 5.3.2 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.

Success Probability: The only randomized parts of the algorithm are the calls of the amplified version of the FPRAS in line 4. Recall that the outer loop is iterated polynomially often, and let p be a polynomial that bounds the number of times the amplified FPRAS is executed. We use an amplified version of the FPRAS where the original FPRAS is called $p(n)$ times, hence the amplified version of the FPRAS has success probability of at least $1 - \frac{1}{2^{p(n)}}$, and by Lemma 3.1.2, it follows that the probability that all calls of the amplified FPRAS succeed is at least $(1 - \frac{1}{2^{p(n)}})^{p(n)} \geq \frac{1}{2}$. Therefore the overall algorithm has success probability of at least $1/2$.

Approximation Ratio: It remains to show that if the algorithm k-`MaxTSPApprox` 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 amplified 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 $3ck$ 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 $3ck \leq |F_H| \leq 3ck^2$ 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 $\beta \in \mathbb{N}^k$ as defined in line 2 and observe that $\beta_i = \min\{w_i(e) \mid e \in F_{H,i}\}$ for all i, which means that for all edges $e \in (R - F_H)$ we have $w(e) \leq \beta$. Hence after line 3 we have $w'(e) = 0$ for all edges $e \in (E - R)$ with $w(e) \not\leq \beta$, and $w'(e) = w(e)$ for all remaining edges e. In particular, it holds that $w'(e) = w(e)$ for all $e \in R$, hence $w'(R) = w(R)$. Further note that w' does not increase the weight of any edge.

Next we obtain a $(1 - 1/|V|)$-approximate set $\mathcal{C}$ of c-cycle covers of (V, E, w') that contain $F_H \cup F_L$. Since R is a c-cycle cover of (V, E, w') with $F_H \cup F_L \subseteq R$, there must be some c-cycle cover $C \in \mathcal{C}$ with $F_H \cup F_L \subseteq C$ that $(1 - 1/|V|)$-approximates R. Hence in line 5 there will be some iteration that chooses this C. Again we fix this iteration for the remainder of the proof.

As in line 6, let $C_1, \dots, C_r$ denote the cycles in C. 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 7 is fulfilled. Let these edges $e_{i,j}$ be defined as in the algorithm.

By the definition of w' and $e_{i,j} \notin F_H$ it holds that $w'(e_{i,j}) \leq \beta$ for all i, j. In line 9 we compute some $\chi\colon \{1, \dots, r\} \to \{1, \dots, c\}$ such that

$$\sum_{i=1}^{r} w'(e_{i,\chi(i)}) \quad \leq \quad 2k \cdot \beta + \frac{1}{c} \sum_{i=1}^{r} \sum_{j=1}^{c} w'(e_{i,j}) \leq 2k \cdot \beta + \frac{1}{c} \cdot w'(C - F_H).$$

Recall that by Corollary 5.3.2 such a coloring exists and can be computed in polynomial time. Removing the chosen edges breaks the cycles into simple paths, which can be arbitrarily connected to a Hamiltonian cycle R'. For the following estimation note that $\beta \leq \frac{w'(F_H)}{3ck}$ and $w'(F_H) \geq \frac{3ck}{m} w'(R)$ and recall that $m = |V| = |R|$. Further recall that $w'(R) = w(R)$. We obtain the following estimation:

$$\begin{aligned}
w(R') &\geq w'(R') \\
&\geq w'(C) - \sum_{i=1}^{r} w'(e_{i,\chi(i)}) \\
&\geq w'(C) - 2k \cdot \beta - \frac{1}{c} \cdot w'(C - F_H) \\
&= \left(1 - \frac{1}{c}\right) w'(C) + \frac{1}{c} w'(F_H) - 2k \cdot \beta \\
&\geq \left(1 - \frac{1}{c}\right) w'(C) + \frac{1}{3c} w'(F_H) \\
&\geq \left(1 - \frac{1}{c}\right) \left(1 - \frac{1}{m}\right) w'(R) + \frac{k}{m} w'(R) \\
&\geq \left(1 - \frac{1}{c}\right) w'(R) \\
&= \left(1 - \frac{1}{c}\right) w(R)
\end{aligned}$$

□

Recall that in directed graphs, every cycle contains at least 2 edges, and in undirected graphs, every cycle contains at least 3 edges. Together with the approximation algorithms from Lemma 5.2.2 we hence obtain the following theorem.

Theorem 5.4.2 *Let $k \geq 1$.*

1. *There exists a randomized $1/2$-approximation algorithm for k-MaxATSP.*
2. *There exists a randomized $2/3$-approximation algorithm for k-MaxSTSP.*

Proof First, consider k-MaxATSP. By Lemma 5.2.2 there exists an FPRAS for k-MaxDFCC$_2$. Hence we can apply Lemma 5.4.1 with $c = 2$ and obtain a randomized $1/2$-approximation algorithm for k-MaxATSP.

Next, consider k-MaxSTSP. Here, by Lemma 5.2.2 there exists an FPRAS for k-MaxUFCC$_3$. Hence we can apply Lemma 5.4.1 with $c = 3$ and obtain a randomized $2/3$-approximation algorithm for k-MaxSTSP. □

5.5 Deterministic Approximation of k-MaxSAT

Our vector balancing results are not restricted in their application to the multiobjective maximum traveling salesman problems. They rather solve a more general problem that frequently appears as a subproblem in multiobjective optimization. Suppose we are given objects with multidimensional weights that are partitioned into disjoint groups, and from each group we need to select one object. Our balancing result shows how to select the objects in a balanced way, such that the overall sum of the weight vectors is balanced in each objective. We will now show a second application of this result to the k-objective maximum weight satisfiability problem k-MaxSAT.

Recall Example 3.1.3, where we describe the following simple approximation algorithm for MaxSAT. For a given formula F in CNF over the set of variables V and a weight function $w\colon F \to \mathbb{N}$, we define $I_0, I_1\colon V \to \{0,1\}$ with $I_0(v) = 0$ and $I_1(v) = 1$ for all $v \in V$ and return I_i with $w(I_i)$ maximal. In Example 3.1.3 we argued that $w(I_0) + w(I_1) \geq w(I)$ for every truth assignment I, and hence the above algorithm computes a $1/2$-approximation for MaxSAT. Again, a trivial adaption to the case of k-MaxSAT fails, because $w(I_0)$ and $w(I_1)$ can be incomparable.

We will describe a different approach, where, for each variable, we define two "influence vectors" on the overall weight of a truth assignment, once when the variable is set to 0, and once when the variable is set to 1. Our balancing result tells us for each variable which influence vector and hence which truth value for the variable we can choose such that the overall weight of the truth assignment thus obtained is roughly $1/2$ of the sum of all influence vectors. Since the sum of the influence vectors will equal the weight of the entire formula, we hence obtain roughly a $1/2$-approximation. Again we have to deal with the small error introduced by the balancing result. Analogously to the case of k-MaxATSP and k-MaxSTSP we can remove this error by fixing constantly many "most influential" variables to their optimal truth value. We now give a detailed description of our algorithm.

For a set of clauses H and a variable v let $H[v = 1] = \{C \in H \mid v \in C\}$ be the set of clauses that are satisfied if this variable is assigned one, and analogously $H[v = 0] = \{C \in H \mid \overline{v} \in C\}$ be the set of clauses that are satisfied if this variable is assigned zero. This notation is extended to sets of variables V by $H[V = i] = \bigcup_{v \in V} H[v = i]$ for $i = 0, 1$. We will first show that for a given $\mathbb{N}^k$-weighted formula in CNF and every truth assignment, there exist $4k^2$ most influential variables such that the weight of the remaining variables can be appropriately bounded.

Lemma 5.5.1 *Let $k \geq 1$, H be a formula in CNF over the variables $V = \{v_1, \ldots, v_m\}$, $w\colon H \to \mathbb{N}^k$, and $I\colon V \to \{0,1\}$. There are sets $V^i \subseteq I^{-1}(\{i\})$, $i = 0, 1$ with $|(V^0 \cup V^1)| \leq 4k^2$ such that for $G = (H - (H[V^0 = 0] \cup H[V^1 = 1]))$ and each $v \in (V - (V^0 \cup V^1))$ it holds that*

$$w(G[v = I(v)]) \leq \frac{1}{4k} w(H - G). \tag{5.1}$$

Proof The lemma obviously holds for $|V| \leq 4k^2$. If $|V| > 4k^2$, we will show the existence of the sets V^0, V^1 by the algorithm `GreedyVariableSelection`.

```
Algorithm: GreedyVariableSelection($H, w, I$)
Input  : Formula $H$ in CNF over the variables $V = \{v_1, \ldots, v_m\}$, $w\colon H \to \mathbb{N}^k$, and truth
         assignment $I\colon V \to \{0,1\}$
Output : Set $\{u_1, \ldots, u_{4k^2}\} \subseteq V$ with $4k$ most influential variables in $I$ per objective
1 $H_0 := H$
2 for $t := 0$ to $4k - 1$ do
3     for $r := 1$ to $k$ do
4         $j := kt + r$
5         choose $v \in (V - \{u_1, \ldots, u_{j-1}\})$ such that $w_r(H_{j-1}[v = I(v)])$ is maximal
6         $u_j := v$
7         $H_j := (H_{j-1} - H_{j-1}[v = I(v)])$
8         $\alpha_j := w(H_{j-1}[v = I(v)])$
9 return $\{u_1, \ldots, u_{4k^2}\}$
```

Let $\{u_1, \ldots, u_{4k^2}\}$ be defined by `GreedyVariableSelection` on input (H, w, I). Note that we use I only to show the existence of V^0, V^1. Let $V^i = I^{-1}(\{i\}) \cap \{u_1, \ldots, u_{4k^2}\}$ for $i = 0, 1$. We now show inequality (5.1), so let $v \in (V - (V^0 \cup V^1))$. Assume that there is some $r \in \{1, \ldots, k\}$ such that $w_r(G[v = I(v)]) > \frac{1}{4k} w_r(H - G)$. Because the union $\bigcup_{j=1}^{4k^2} H_{j-1}[u_j = I(u_j)] = (H - G)$ is disjoint, we get

$$w(H - G) = \sum_{r'=1}^{k} \sum_{t=0}^{4k-1} \alpha_{kt+r'} \geq \sum_{t=0}^{4k-1} \alpha_{kt+r}$$

and thus

$$4k \cdot w_r(G[v = I(v)]) > \sum_{t=0}^{4k-1} (\alpha_{kt+r})_r \ .$$

Hence, by the pigeonhole principle, there must be some $t \in \{0, 1, \ldots, 4k-1\}$ such that $w_r(G[v = I(v)]) > (\alpha_{kt+r})_r$. But since $G \subseteq H_{kt+r-1}$ and thus even $w_r(H_{kt+r-1}[v = I(v)]) \geq w_r(G[v = I(v)]) > (\alpha_{kt+r})_r$, the variable v should have been chosen in step $j = kt + r$, which is a contradiction. This means that $w(G[v = I(v)]) \leq \frac{1}{4k} w(H - G)$ holds for all $v \in (V - (V^0 \cup V^1))$. □

We will now describe the approximation algorithm for k-MaxSAT, where $k \geq 1$.

Algorithm: k-`MaxSATApprox`(H, w)

```
   Input   : Formula $H$ in CNF over the variables $V = \{v_1, \ldots, v_m\}$, and $w\colon H \to \mathbb{N}^k$
   Output  : Truth assignments $I\colon V \to \{0,1\}$
1  foreach disjoint $V^0, V^1 \subseteq V$ with $|(V^0 \cup V^1)| \leq 4k^2$ do
2      $G := (H - (H[V^0 = 0] \cup H[V^1 = 1]))$
3      $\hat{V}^{(1-i)} := \{v \in (V - (V^0 \cup V^1)) \mid 4k \cdot w(G[v = i]) \not\leq w(H - G)\}$, $i = 0, 1$
4      if $\hat{V}^0 \cap \hat{V}^1 = \emptyset$ then
5          $V' := (V - (V^0 \cup V^1 \cup \hat{V}^0 \cup \hat{V}^1))$, $L' := V' \cup \{\overline{v} \mid v \in V'\}$
6          $G' := ((G[V' = 0] \cup G[V' = 1]) - (G[\hat{V}^0 = 0] \cup G[\hat{V}^1 = 1]))$
7          for $v_j \in V'$ let $x^{j,i} = \sum \left\{ \frac{w(C)}{|(C \cap L')|} \mid C \in G'[v_j = i] \right\}$ for $i = 0, 1$
8          compute some coloring $\chi\colon V' \to \{0,1\}$ such that

               $$\sum_{v_j \in V'} x^{j,\chi(j)} \geq \frac{1}{2} \sum_{v_j \in V'} (x^{j,0} + x^{j,1}) - 2k\delta$$

           where $\delta \in \mathbb{N}^k$ such that $\delta_r = \max\{x_r^{j,i} \mid v_j \in V', i \in \{0,1\}\}$
9          let $I(v) := i$ for $v \in V^i \cup \hat{V}^i \cup \chi^{-1}(\{i\})$, $i = 0, 1$
10         output $I$
```

Theorem 5.5.2 *For every $k \geq 1$, k-MaxSAT is $1/2$-approximable.*

Proof We show that this approximation is realized by k-`MaxSATApprox`. So let (H, w) be the input, where H is a formula in CNF over the variables $V = \{v_1, \ldots, v_m\}$ and $w\colon H \to \mathbb{N}^k$ is the k-objective weight function.

The algorithm only returns truth assignments of V and hence works correctly. We argue for polynomial runtime and the claimed approximation ratio.

Runtime: Since k is a constant, there are only polynomially many disjoint sets $V_0, V_1 \subseteq V$ with $|(V_0 \cup V_1)| \leq 4k^2$. So there are only polynomially many iterations of the algorithm. The coloring in line 8 can be computed in polynomial time using Corollary 5.3.2, while all remaining steps are trivial. So overall we obtain a polynomial-time algorithm.

Approximation Ratio: Let $I_o\colon V \to \{0,1\}$ be an arbitrary truth assignment. We show that there is an iteration of k-`MaxSATApprox`(H, w) that outputs a truth assignment I such that $w(I) \geq {}^{w(I_o)}/_2$.

We first note that there are sets V^0 and V^1 with a bounded cardinality of at most $4k^2$ that define a partial truth assignment that contributes a large weight.

We choose the iteration of the algorithm k-`MaxSATApprox` where V^0 and V^1 equal the sets whose existence is guaranteed by Lemma 5.5.1. In the following, we use the variables as they are defined in the algorithm. Observe that by Lemma 5.5.1 it holds that $I_o(v) = i$ for all $v \in \hat{V}^i$ for $i = 0, 1$ and thus $\hat{V}^0 \cap \hat{V}^1 = \emptyset$. Note that

$$\sum_{v_j \in V'} (x^{j,0} + x^{j,1}) = \sum_{v_j \in V'} \sum_{i \in \{0,1\}} \sum_{C \in G'[v_j = i]} \frac{w(C)}{|(C \cap L')|} = \sum_{C \in G'} |(C \cap L')| \frac{w(C)}{|(C \cap L')|} = w(G').$$

Furthermore, for all $v_j \in V'$ and $i = 0, 1$, we have the bound $x^{j,i} \leq w(G'[v_j = i]) \leq w(G[v_j = i]) \leq \frac{1}{4k} w(H - G)$ because of the definition of V' and $\hat{V}^{(1-i)}$. By Corollary 5.3.2, we find a coloring $\chi\colon V' \to \{0,1\}$ such that for each $1 \leq i \leq k$ it holds that

$$\operatorname{abs}\left(\frac{1}{2} \sum_{v_j \in V'} \sum_{r=0}^{1} x_i^{j,r} - \sum_{v_j \in V'} x_i^{j,\chi(v_j)}\right) \leq 2k \max_{j,r} \operatorname{abs}(x_i^{j,r}) \leq 2k \frac{1}{4k} w_i(H - G) = \frac{1}{2} w_i(H - G)$$

and hence

$$\sum_{v_j \in V'} x^{j,\chi(v_j)} \geq \frac{1}{2} \sum_{v_j \in V'} (x^{j,0} + x^{j,1}) - \frac{1}{2} w(H - G) = \frac{1}{2}(w(G') - w(H - G)).$$

For I being the truth assignment generated in this iteration it holds that

$$w(\{C \in G' \mid I(C) = 1\}) \geq \sum_{v_j \in V'} x^{j,\chi(v_j)} \geq \frac{1}{2}(w(G') - w(H - G)). \tag{5.2}$$

Furthermore, since I and I_o coincide on $(V - V')$, we have

$$\begin{aligned} w(\{C \in (H - G') \mid I(C) = 1\}) &= w(\{C \in (H - G') \mid I_o(C) = 1\}) \qquad (5.3)\\ &\geq w(\{C \in (H - G) \mid I_o(C) = 1\}) \\ &= w(H - G). \qquad (5.4) \end{aligned}$$

Thus we obtain

$$\begin{aligned} w(I) &= w(\{C \in (H - G') \mid I(C) = 1\}) + w(\{C \in G' \mid I(C) = 1\}) \\ &\geq w(\{C \in (H - G') \mid I(C) = 1\}) + \tfrac{1}{2}(w(G') - w(H - G)) && \text{(by (5.2))} \\ &= w(\{C \in (H - G') \mid I_o(C) = 1\}) + \tfrac{1}{2}(w(G') - w(H - G)) && \text{(by (5.3))} \\ &\geq \tfrac{1}{2} w(\{C \in (H - G') \mid I_o(C) = 1\}) + \tfrac{1}{2} w(G') && \text{(by (5.4))} \\ &\geq \tfrac{1}{2} w(I_o). \end{aligned}$$

□

5.6 Summary and Discussion

In this chapter we shifted our attention to multiobjective maximization variants of the traveling salesman problem. We distinguished between the version on directed graphs and the version on undirected graphs, where we obtained a randomized $1/2$ and a randomized $2/3$ approximation algorithm, respectively. Both algorithms improve the previously best known approximation algorithms, which are due to Manthey [Man12b], by removing the ε-error in the approximation ratio.

In order to obtain these approximation algorithms, we considered typical single-objective algorithms that are based on cycle covers and generalized them to the multiobjective setting. Such a generalization is not trivial, which is due to a subproblem that frequently occurs in multiobjective optimization: elements with multiobjective weights can be incomparable, and terms like “lightest edge” are not well-defined if the weights have multiple components, so we cannot simply remove the “lightest edge” but need more sophisticated techniques.

We showed that results from discrepancy theory yield a solution to this subproblem. Corollary 5.3.2 provides us with a polynomial-time algorithm that selects multidimensional weight vectors from a list of such vectors such that some selection constraints are satisfied and such that the selected vectors have a weight that is balanced (i.e., that roughly equals a fixed fraction of the weight of all vectors) in each objective. This subproblem is very general and common in multiobjective optimization, hence Corollary 5.3.2 is of particular interest here. As another application we used it to find an approximation algorithm for a multiobjective version of the maximum satisfiability problem.

Chapter 6

Applications of Necklace Splitting

Deterministic Approximation of Multiobjective Maximum Traveling Salesman Recall from the last chapter that there exists a randomized $1/2$-approximation algorithm for k-MaxATSP and a randomized $2/3$-approximation algorithm for k-MaxSTSP, where $k \geq 1$. These algorithms compute cycle covers by a reduction to the multiobjective maximum perfect matching problem, for which randomized approximation algorithms are known by Theorem 3.2.1. Hence the algorithms for k-MaxATSP and k-MaxSTSP are randomized.

Manthey [Man12a] was the first to study the deterministic approximability of k-MaxATSP and k-MaxSTSP. Since no FPTAS is known for the multiobjective maximum perfect matching problems, it becomes more difficult to find a deterministic approximation for k-MaxATSP and k-MaxSTSP. Manthey [Man12a] showed that k-MaxATSP is deterministically $(1-\varepsilon)/(4k-2)$-approximable, and k-MaxSTSP is deterministically $(1-\varepsilon)/2k$-approximable, where $k \geq 2$ and $\varepsilon > 0$. Moreover, in the two-objective case, he obtained a deterministic $(1-\varepsilon)/4$-approximation algorithm for 2-MaxATSP and a deterministic $(3-\varepsilon)/8$-approximation algorithm for 2-MaxSTSP.

Necklace Splitting in Multiobjective Approximation We use cycle cover based algorithms to deterministically approximate the multiobjective traveling salesman problems. In the last chapter we showed how to deterministically transform a cycle cover into a Hamiltonian cycle without losing too much weight in every objective. So it remains to find a good cycle cover approximation, which again reduces to multiobjective matching problems. Since we want to find deterministic algorithms, we will not use the randomized approximation scheme for multiobjective matching. Instead we will use necklace splitting results to deterministically obtain multiobjective matching approximations.

For a given graph $G = (V, E, w)$ with $w\colon E \to \mathbb{N}^k$, we will consider the *(perfect) matching polytope*, which is defined as the convex hull of incidence vectors of (perfect) matchings of G. We will show that for every $d \in \mathbb{N}^k$ we can either find a point $x \in \mathbb{R}^E$ in the matching polytope that corresponds to a *fractional matching* M with $w(M) \geq d$, or we obtain the assertion that no such fractional matching exists. Given a fractional matching M, we can use an algorithmic version of Carathéodory's Theorem [Car11, GLS88] to compute a constant number of matchings $M_1, \ldots, M_l$ such that M is a convex combination of $M_1, \ldots, M_l$. It remains to combine $M_1, \ldots, M_l$ into a single matching M' whose weight roughly equals the weight of the fractional matching M.

At this point we will apply solutions to the *necklace splitting problem.* In its original problem formulation, two thieves want to split a valuable necklace into two halves such that each of them obtains exactly one half of the material and hence of the value of the necklace. The task is to use as few cuts as possible. Clearly, if the necklace consists of a single material, then two cuts are sufficient to split the necklace evenly. The question becomes more interesting if one considers necklaces that consist of multiple, different materials. A result by Stromquist and Woodall [SW85] states that if we are given a necklace made of $k \geq 2$ different materials and some $\alpha \in [0, 1]$, then $2(k-1)$ cuts are sufficient to split the necklace in such a way that one can remove a fraction of α of each material. We apply these necklace splitting results and obtain that if the fractional matching M is a convex combination of the matchings $M_1, \ldots, M_l$ with known coefficients, then we can extract a matching M' from $M_1, \ldots, M_l$ that has almost the same weight as M in every objective. The few cuts of the necklace splitting result make sure that the difference between the weights of M and M' is small. Moreover, we can show that if $M_1, \ldots, M_l$ are perfect, then M' is r-near perfect, where r is relatively small.

Since now we can decide for every $d \in \mathbb{N}^k$ whether there exists a matching at least as good as d in every objective, and we can further compute a matching that is almost as good as d if a matching at least as good as d exists, we can apply a method by Papadimitriou and Yannakakis [PY00] and obtain a PTAS for the multiobjective maximum matching problem. We remark at this point that recently, Chekuri et al. [CVZ11] have shown that there exists a PTAS for multiobjective maximum matching. They work with fractional solutions in the matching polytope and use *randomized rounding* to obtain a good approximate solution. While our approach also works on the matching polytope, we use necklace splitting to combine multiple solutions into a good approximation.

Since we can make sure that perfect matchings are combined into r-near perfect matchings for some small r, we also obtain a PTAS-like approximation algorithm that approximates perfect matchings by r-near perfect matchings. By a number of reductions we show that the multiobjective cycle cover problems on directed and on undirected graphs admit PTAS-like approximation algorithms, which we will then use to obtain deterministic approximation algorithms for k-MaxATSP and k-MaxSTSP.

Contributions We summarize the contributions of this chapter as follows.

1. *A Structured Vector Balancing Tool.* We apply the necklace splitting results by Stromquist and Woodall [SW85] and obtain a balancing result for vectors that is similar to the result of the last chapter, where for a set of given vectors we obtained a subset that was almost balanced in each component. While the result in this chapter has a bigger deviation from an optimal balance, it is more structured. Namely, for every *list* of vectors, we can find an almost balanced coloring of the vectors that only has a constant number of jump points. This in particular shows that the coloring can be found in polynomial time by brute force.
2. *Improved Approximations.* Using the structured balancing tool we provide a PTAS for the multiobjective maximum matching problem (k-MaxM) and a PTAS-like approximation algorithm for the multiobjective maximum perfect matching problem (k-MaxPM) that approximates perfect matchings by near-perfect matchings. Note that by Chekuri et al. [CVZ11], a PTAS for the multiobjective maximum matching problem exists. Our approach uses necklace splitting and can also be used to approximate perfect matchings by l-near perfect matchings for some small l. By a number of reductions we obtain a PTAS for the multiobjective maximum partial cycle cover problem on directed graphs, where up to r edges can be fixed (k-r-MaxDFPCC), and a PTAS-like approximation algorithm for the multiobjective maximum cycle cover problem on undirected graphs, where up to r edges can be fixed (k-r-MaxUFCC), that approximates cycle covers by l-near cycle covers. As the main result of this chapter we obtain that there exist a deterministic $^{(1-\varepsilon)}/_{2}$-approximation algorithm for k-MaxATSP and a deterministic $^{(2-\varepsilon)}/_{3}$-approximation algorithm for k-MaxSTSP. We summarize our approximation results in Table 6.1.

Problem	Approximation	Reference	Remark
k-MaxM	PTAS	6.3.11	
k-MaxPM	PTAS-like	6.3.12	algorithm outputs $8k^2$-near perfect matchings
k-r-MaxDFPCC	PTAS	6.4.1	
k-r-MaxUFCC	PTAS-like	6.4.5	algorithm outputs $8k^2$-near cycle covers
k-MaxATSP	$(1-\varepsilon)/2$	6.5.2	
k-MaxSTSP	$(2-\varepsilon)/3$	6.5.2	

Table 6.1: Deterministic approximation ratios shown in this chapter, where $\varepsilon > 0$ and $k \geq 1$, $r \geq 0$.

Organization of this Chapter In Section 6.1 we define the problems we consider in this chapter. In Section 6.2 we apply the necklace splitting results from the literature to obtain a structured vector balancing tool. We continue to study deterministic approximation algorithms for the multiobjective matching problems in Section 6.3. In Section 6.4 we will reduce multiobjective cycle cover problems to the multiobjective matching problems. By the results of Section 6.3 we will obtain PTAS-like approximation algorithms for the cycle cover problems. We apply the cycle cover approximation algorithms in Section 6.5 to show how to obtain deterministic approximation algorithms for k-MaxATSP and k-MaxSTSP. Our results are summarized in Section 6.6.

6.1 Problem Definitions

Recall the notions of a *r-near perfect matching* and a *r-near cycle cover* from Section 2.2. Let $k \geq 1$ and further recall the definition of k-MaxATSP and k-MaxSTSP from the previous chapter. We will develop deterministic approximation algorithms for k-MaxATSP and k-MaxSTSP using reductions to cycle cover problems, which can be reduced to matching problems. We consider the following multiobjective matching problems.

k-Objective Maximum Weight Perfect Matching

Notation:	k-MaxPM
Instance:	$\mathbb{N}^k$-labeled undirected graph $G = (V, E, w)$
Solution:	perfect matching $M \subseteq E$ of G
Objective:	maximize $w(M)$

k-Objective Maximum Weight Matching

Notation:	k-MaxM
Instance:	$\mathbb{N}^k$-labeled undirected graph $G = (V, E, w)$
Solution:	matching $M \subseteq E$ of G
Objective:	maximize $w(M)$

We show that k-MaxM admits a PTAS and that k-MaxPM admits a PTAS-like approximation that outputs $8k^2$-near perfect matchings instead of perfect matchings. Using these algorithms, we will show approximation algorithms for the following cycle cover problems.

k-Objective Maximum Directed r-Fixed Partial Cycle Cover

Notation:	k-r-MaxDFPCC
Instance:	$\mathbb{N}^k$-labeled directed graph (V, E, w) and $F \subseteq E$ with $\lvert F\rvert \leq r$
Solution:	partial cycle cover $C \subseteq E$ with $F \subseteq C$
Objective:	maximize $w(C)$

k-Objective Maximum Undirected r-Fixed Cycle Cover
Notation: k-r-MaxUFCC
Instance: $\mathbb{N}^k$-labeled undirected graph (V, E, w) and $F \subseteq E$ with $|F| \leq r$
Solution: cycle cover $C \subseteq E$ with $F \subseteq C$
Objective: maximize $w(C)$

6.2 Necklace Splitting Results

Stromquist and Woodall [SW85] use a general form of the so-called ham sandwich theorem by Stone and Tukey [ST42] to show that, given k probability measures on the unit circle S^1, for every portion $\alpha \in [0, 1]$ there exists a union of $k-1$ intervals from the unit circle whose measures all equal α.

Theorem 6.2.1 ([SW85]) *Let $n \geq 2$ and let $\mu_1, \ldots, \mu_n$ be non-atomic probability measures on S^1. For each $\alpha \in [0, 1]$ there is a set $K_\alpha \subseteq S^1$ such that K_α is a union of at most $n-1$ intervals and $\mu_i(K_\alpha) = \alpha\mu_i(S^1) = \alpha$ for each i ($i = 1, \ldots, n$).*

Corollary 6.2.2 *Let $k \geq 2$ and $f_1, \ldots, f_k \colon [0, 1] \to \mathbb{R}$ be integrable and non-negative. For every $\alpha \in [0, 1]$ there is some coloring $\chi \colon [0, 1] \to \{0, 1\}$ with at most $2(k-1)$ jumps such that for all $i \in \{1, \ldots, k\}$,*

$$\int_{\chi^{-1}(1)} f_i(x)dx = \alpha \int_{[0,1]} f_i(x)dx.$$

Proof We assume that for each i it holds that $\int_{[0,1]} f_i(x)dx > 0$, otherwise the equation in Corollary 6.2.2 trivially holds for f_i. For each i we define a non-atomic probability measure μ_i on S^1 by

$$\mu_i(R) := \frac{\int_R f(x)dx}{\int_{[0,1]} f(x)dx}$$

for all $R \subseteq S^1$. From Theorem 6.2.1 we obtain a set $K_\alpha \subseteq S^1$ such that K_α is a union of at most $k-1$ intervals and $\mu_i(K_\alpha) = \alpha$ for all i. So for the coloring $\chi \colon [0, 1] \to \{0, 1\}$ with $\chi(x) = 1 \iff x \in K_\alpha$ we obtain

$$\int_{\chi^{-1}(1)} f_i(x)dx = \int_{K_\alpha} f_i(x)dx = \mu_i(K_\alpha) \cdot \int_{[0,1]} f_i(x)dx = \alpha \int_{[0,1]} f_i(x)dx$$

for each i. Since K_α is a union of at most $k-1$ intervals, the coloring χ has at most $2(k-1)$ jumps. □

Corollary 6.2.3 *Let $c, k \geq 1$ and for $r = 1, \ldots, c$ and $i = 1, \ldots, k$ let $f_{r,i} \colon [0, 1] \to \mathbb{R}$ be integrable and non-negative, and let $\alpha \in [0, 1]^c$ such that $\|\alpha\|_1 = 1$. There is a coloring $\chi \colon [0, 1] \to \{1, \ldots, c\}$ with at most $2kc^2$ jumps such that for all $i \in \{1, \ldots, k\}$,*

$$\int_{[0,1]} f_{\chi(x),i}(x)dx = \sum_{r=1}^{c} \alpha_r \cdot \int_{[0,1]} f_{r,i}(x)dx.$$

Proof We show this by induction on c. The case that $\alpha_c = 1$ is trivial, which includes the induction base $c = 1$. For the induction step, we ignore the grouping of the $c \cdot k$ functions and obtain from Corollary 6.2.2 some $\chi'\colon [0,1] \to \{0,1\}$ with at most $2(ck-1)$ jumps such that for all r, i,

$$\int_{\chi'^{-1}(1)} f_{r,i}(x)dx = \alpha_c \int_{[0,1]} f_{r,i}(x)dx.$$

Now define $f'_{r,i}(x) = (1-\chi'(x)) \cdot f_{r,i}(x)$. The induction hypothesis applied to $f'_{r,i}$, $1 \le r \le c-1$, $1 \le i \le k$ and the vector $\alpha' \in [0,1]^{c-1}$, $\alpha'_r = \frac{\alpha_r}{1-\alpha_c}$ yields a coloring $\chi''\colon [0,1] \to \{1,\dots,c-1\}$ with at most $2k(c-1)^2$ jumps such that for all $i \in \{1,\dots,k\}$,

$$\int_{[0,1]} f'_{\chi''(x),i}(x)dx = \sum_{r=1}^{c-1} \alpha'_r \cdot \int_{[0,1]} f'_{r,i}(x)dx.$$

For

$$\chi(x) = \begin{cases} c & \text{if } \chi'(x) = 1 \text{, and} \\ \chi''(x) & \text{otherwise} \end{cases}$$

we obtain for all $1 \le i \le k$,

$$\begin{aligned}
\int_{[0,1]} f_{\chi(x),i}(x)dx &= \int_{\chi'^{-1}(1)} f_{c,i}(x)dx + \int_{\chi'^{-1}(0)} f_{\chi''(x),i}(x)dx \\
&= \int_{\chi'^{-1}(1)} f_{c,i}(x)dx + \int_{[0,1]} f'_{\chi''(x),i}(x)dx \\
&= \alpha_c \int_{[0,1]} f_{c,i}(x)dx + \sum_{r=1}^{c-1} \alpha'_r \cdot \int_{[0,1]} f'_{r,i}(x)dx \\
&= \alpha_c \int_{[0,1]} f_{c,i}(x)dx + \sum_{r=1}^{c-1} \alpha'_r \cdot (1-\alpha_c) \int_{[0,1]} f_{r,i}(x)dx \\
&= \sum_{r=1}^{c} \alpha_r \int_{[0,1]} f_{r,i}(x)dx.
\end{aligned}$$

The number of jumps of χ is $2(ck-1) + 2k(c-1)^2 \le 2kc^2$. □

Corollary 6.2.4 *Let $c, k \ge 1$ and $n \ge 0$, and let $\alpha \in [0,1]^c$ such that $\|\alpha\|_1 = 1$. For each set of vectors $v^{j,r} \in \mathbb{R}^k$ with non-negative components for $j = 1,\dots,n$ and $r = 1,\dots,c$, there is a coloring $\chi\colon \{1,\dots,n\} \to \{1,\dots,c\}$ with at most $2kc^2$ jumps such that for each $1 \le i \le k$ it holds that*

$$\operatorname{abs}\left(\sum_{j=1}^{n} v_i^{j,\chi(j)} - \sum_{r=1}^{c} \alpha_r \sum_{j=1}^{n} v_i^{j,r}\right) \le 2kc^2 \max_{j,r} v_i^{j,r}.$$

Proof For each $i = 1,\dots,k$ and $r = 1,\dots,c$ define the function $f_{r,i}\colon [0,1] \to \mathbb{R}$ with $f_{r,i}(x) = n \cdot v_i^{\lceil xn \rceil, r}$ and $f_{r,i}(0) = 0$. From Corollary 6.2.3, we obtain a coloring $\chi'\colon [0,1] \to \{1,\dots,c\}$ with at most $2kc^2$ jumps such that for all $i \in \{1,\dots,k\}$

$$\int_{[0,1]} f_{\chi'(x),i}(x)dx = \sum_{r=1}^{c} \alpha_r \cdot \int_{[0,1]} f_{r,i}(x)dx = \sum_{r=1}^{c} \alpha_r \sum_{j=1}^{n} v_i^{j,r}.$$

Define $\chi''\colon [0,1] \to \{1,\ldots,c\}$ from χ' by moving each jump point at $x \in [0,1]$ to the next value such that $n \cdot x$ is an integer. Since we have to move at most $2kc^2$ jump points each over a distance of at most $\frac{1}{n}$, we get for $i \in \{1,\ldots,k\}$

$$\operatorname{abs}\left(\int_{[0,1]} f_{\chi'(x),i}(x)dx - \int_{[0,1]} f_{\chi''(x),i}(x)dx\right) \leq 2kc^2 \max_{j,r} v_i^{j,r}.$$

The assertion is obtained for the coloring $\chi\colon \{1,\ldots,n\} \to \{1,\ldots,c\}$, $\chi(j) := \chi''(\frac{j-\frac{1}{2}}{n})$. □

Note that Corollary 5.3.2 and Corollary 6.2.4 are very similar. However, while Corollary 5.3.2 has a smaller error factor, Corollary 6.2.4 guarantees a constant number of jumps. This will be crucial in the remainder of this chapter.

6.3 Deterministic Matching Approximation

We next show that there exists a PTAS for the multiobjective maximum matching problem and a PTAS-like approximation algorithm for the multiobjective maximum perfect matching problem. The PTAS-like algorithm has the same approximation guarantee as a PTAS but approximates perfect matchings only by r-near perfect matchings for some small r. Let $G = (V, E, w)$ be an $\mathbb{N}^k$-labeled undirected graph. We will proceed as follows.

1. We first show that for every $d \in \mathbb{N}^k$ we can find out whether there exists a *fractional matching* M (i.e., a point in the matching polytope) with $w(M) \geq d$ or not. Moreover, if there exists such a fractional matching, then we can compute it in polynomial time.
2. We next argue that the fractional matching M is a convex combination of constantly many matchings $M_0, \ldots, M_k$, which can be computed from M and G in polynomial time.
3. We proceed to prove that in polynomial time we can combine $M_0, \ldots, M_k$ to a single matching M' such that the difference between $w(M')$ and $w(M)$ is relatively small. The combination of different matchings into a single matching uses the necklace splitting results from the last section and only introduces a relatively small error.
4. For a given $\varepsilon > 0$ we can repeat the above steps at polynomially many points $d \in \mathbb{N}^k$ and obtain a set of matchings such that for every matching M^* of G there exists a matching in our set that $(1-\varepsilon)$-approximates M^*.

Notations and Definitions Let E be a set. For convenience, in this section we will adapt the notation that $\mathbb{R}^E$ is the set of mappings from E to $\mathbb{R}$, where the components of an element $x \in \mathbb{R}^E$ are indexed by the elements $e \in E$, hence $x = (x_e)_{e \in E}$. Let $M \subseteq E$. A vector $x \in \mathbb{R}^E$ is called *incidence vector of* M if and only if $x_e = 1$ if $e \in M$ and $x_e = 0$ if $e \notin M$ for all $e \in E$.

Let $n, k \in \mathbb{N}$. We say that a vector $x \in \mathbb{R}^n$ is a *linear combination* of the vectors $y_1, \ldots, y_k \in \mathbb{R}^n$ if there exists some $\lambda \in \mathbb{R}^k$ such that $x = \sum_{i=1}^k \lambda_i y_i$. A linear combination with $\sum_{i=1}^k \lambda_i = 1$ is called *affine*, and an affine combination with $\lambda_i \geq 0$ for all $1 \leq i \leq k$ is called *convex*.

Let $P \subseteq \mathbb{R}^n$ be a set. We define the *convex hull of* P by

$$\operatorname{conv}(P) := \{x \in \mathbb{R}^n \mid x \text{ is a convex combination of some } y_1, \ldots, y_m \in P \text{ with } m \in \mathbb{N}\}.$$

P is called *affinely independent* if there do not exist distinct $x, y_1, \ldots, y_m \in P$ with $m \geq 1$ such that x is an affine combination of $y_1, \ldots, y_m$. The *affine rank of* P is defined by

$$\operatorname{arank}(P) := \max\{|S| \mid S \subseteq P \text{ and } S \text{ is affinely independent}\}.$$

It is well-known that from $P \subseteq \mathbb{R}^n$ it follows that $\operatorname{arank}(P) \leq n+1$. If $\operatorname{arank}(P) = n+1$, then P is called *full-dimensional.* P is called a *polyhedron* if $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$ for some $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$, and if it further holds that P is contained in a sphere in $\mathbb{R}^n$, then P is called a *polytope.* If P is a polyhedron and $x \in P$, we call x a *vertex of* P if x is not a convex combination of other elements of P.

Matching Polytopes Let $G = (V, E)$ be some undirected graph.

$$M(G) := \operatorname{conv}(\{x \in \mathbb{R}^E \mid x \text{ is incidence vector of a matching in } G\})$$
$$P(G) := \operatorname{conv}(\{x \in \mathbb{R}^E \mid x \text{ is incidence vector of a perfect matching in } G\})$$

We call $M(G)$ the *matching polytope of* G and $P(G)$ the *perfect matching polytope of* G. Clearly it holds that $P(G) \subseteq M(G)$. Note that for all $x \in \mathbb{R}^E$ it holds that x is a vertex of $M(G)$ if and only if x is the incidence vector of a matching of G, and x is a vertex of $P(G)$ if and only if x is the incidence vector of a perfect matching of G. Observe that the set $\{x \in \{0,1\}^E \mid \sum_{e \in E} x_e \leq 1\} \subseteq M(G)$ is affinely independent and has cardinality $|E| + 1$, hence $M(G)$ is full-dimensional.

Edmonds [Edm65] gave a characterization of the matching polytopes by a system of inequalities (see Schrijver [Sch83] for an alternative proof).

Theorem 6.3.1 ([Edm65, Sch83]) *Let $G = (V, E)$ be an undirected graph.*

1. *$M(G) = \{x \in \mathbb{R}^E \mid x$ satisfies (6.1) - (6.3)$\}$, where:*

$$x_e \geq 0 \qquad \textit{(for each } e \in E\textit{)} \tag{6.1}$$
$$\sum_{e \in E:\, v \in e} x_e \leq 1 \qquad \textit{(for each } v \in V\textit{)} \tag{6.2}$$
$$\sum_{e \in E:\, e \subseteq U} x_e \leq \frac{|U| - 1}{2} \qquad \textit{(for each } U \subseteq V \textit{ with } |U| \textit{ odd)} \tag{6.3}$$

2. *$P(G) = \{x \in \mathbb{R}^E \mid x$ satisfies (6.4) - (6.6)$\}$, where:*

$$x_e \geq 0 \qquad \textit{(for each } e \in E\textit{)} \tag{6.4}$$
$$\sum_{e \in E:\, v \in e} x_e = 1 \qquad \textit{(for each } v \in V\textit{)} \tag{6.5}$$
$$\sum_{e \in E:\, |e \cap U| = 1} x_e \geq 1 \qquad \textit{(for each } U \subseteq V \textit{ with } |U| \textit{ odd)} \tag{6.6}$$

Separation Problems for the Matching Polytopes Let $P \subseteq \mathbb{R}^n$ be some polytope. Following Grötschel, Lovász and Schrijver [GLS81], we define separation problems for P as follows.

> **Strong Separation Problem for P:**
> Given a vector $y \in \mathbb{R}^n$, either assert that $y \in P$, or find a vector $c \in \mathbb{R}^n$ such that $c^T y > c^T x$ for all $x \in P$.

We can hence think of $\{x \in \mathbb{R}^n \mid c^T x = c^T y\}$ as a hyperplane in $\mathbb{R}^n$ that separates y and P. Without any further assumptions, it might be possible that c is not a vector with rational components, hence we also introduce a weak version that allows a small deviation and makes sure that for every separating hyperplane there exists a separating hyperplane with rational c.

Weak Separation Problem for P:
Given a vector $y \in \mathbb{Q}^n$ and $\varepsilon > 0$, either assert that $d(y, P) \leq \varepsilon$, or find a vector $c \in \mathbb{Q}^n$ with $\|c\|_1 \geq 1$ such that $c^T y + \varepsilon \geq c^T x$ for all $x \in P$.

Here, $d(y, P)$ denotes the Euclidean distance of y from P. It is known that there exist polynomial-time algorithms that solve the separation problem for the matching polytope and for the perfect matching polytope. For the sake of completeness we give a short proof.

Theorem 6.3.2 ([GLS81, PR82]) *There are polynomial-time algorithms that on input of an undirected graph $G = (V, E)$ and $y \in \mathbb{Q}^E$ solve the strong separation problems (on rational inputs) and hence also the weak separation problems for $M(G)$ and $P(G)$, respectively.*

Proof Suppose we have the input $G = (V, E)$ and $y \in \mathbb{Q}^E$. Let us first argue for the perfect matching polytope. Recall that $P(G)$ is the set of all solutions to the inequalities from (6.4) - (6.6). We can easily check whether y satisfies the inequalities from (6.4) and (6.5), where a violation immediately yields a hyperplane separating y and $P(G)$. So it remains to check whether the inequalities from (6.6) hold for y, i.e., whether it holds that

$$\sum_{e \in E:\ |e \cap U| = 1} y_e \geq 1 \qquad \text{(for each } U \subseteq V \text{ with } |U| \text{ odd).}$$

This test is non-trivial, because there are exponentially many subsets $U \subseteq V$. However, Padberg and Rao [PR82] showed how to perform this test in polynomial time. They argue that finding a set $U' \subseteq V$ such that $|U'|$ is odd and $\sum_{e \in E:\ |e \cap U'| = 1} y_e$ is minimal can be done in polynomial time by a reduction to a polynomial-time solvable minimum odd cut problem. Now we have the following cases.

- $\sum_{e \in E:\ |e \cap U'| = 1} y_e < 1$. Then U' yields a violation of (6.6) and hence a separating hyperplane.
- $\sum_{e \in E:\ |e \cap U'| = 1} y_e \geq 1$. Then $1 \leq \sum_{e \in E:\ |e \cap U'| = 1} y_e \leq \sum_{e \in E:\ |e \cap U| = 1} y_e$ holds for all $U \subseteq V$ with $|U|$ odd, so U' asserts that (6.6) also holds for y.

The matching polytope $M(G)$ is treated in a similar fashion. We can again trivially check (6.1) and (6.2), while Padberg and Rao [PR82] show how to reduce (6.3) to (6.6) by introducing slack variables. □

Corollary 6.3.3 *Let $k \in \mathbb{N}$. There are polynomial-time algorithms that on input of an $\mathbb{N}^k$-labeled, undirected graph $G = (V, E, w)$ and $d \in \mathbb{N}^k$ perform as follows.*

1. *Return some $x \in M(G)$ such that $w(x) \geq d$, or correctly state that no such x exists.*
2. *Return some $x \in P(G)$ such that $w(x) \geq d$, or correctly state that no such x exists.*

Proof We begin with the first item. For the input $G = (V, E, w)$ and $d \in \mathbb{N}^k$ we define the polytope

$$M'(G) = M(G) \cap \{x \in \mathbb{R}^E \mid w(x) \geq d\}.$$

Recall that by Theorem 6.3.2, the strong separation problem for $M(G)$ (on rational inputs) can be solved in polynomial time, and observe that the strong separation problem for $\{x \in \mathbb{R}^E \mid w(x) \geq d\}$ (on rational inputs) can trivially be solved in polynomial time by checking each inequality, where a violation yields a separating hyperplane. Hence the strong separation problem for $M'(G)$ (on rational inputs) can be solved in polynomial time.

Plummer and Lovász [PL86] state that for classes of “well-behaved polytopes”, if the strong separation problem on rational inputs can be solved in polynomial time, then the ellipsoid method (see Grötschel, Lovász and Schrijver [GLS88]) can be used to maximize arbitrary linear functions over each contained polytope in polynomial time. In this context, “well-behaved” means that the dimension of $M'(G)$ is polynomial in the input, and moreover, the vertices of $M'(G)$ are vectors of rational numbers whose numerator and denominator can be described with at most polynomially many bits. Observe that this is satisfied by $M'(G)$, hence we can optimize an arbitrary linear function over $M'(G)$ and either obtain some $x \in M'(G)$ or the assertion that $M'(G)$ is empty. If $M'(G)$ is empty, then $x \in M(G)$ implies $w(x) \not\geq d$ and hence there is no $x \in M(G)$ such that $w(x) \geq d$. Otherwise we obtain some $x \in M'(G)$. In this case it holds that $x \in M(G)$ and $w(x) \geq d$, and we are done.

The second item is shown analogously, where we consider the polytope $P(G)$ instead. □

Computations in the Matching Polytope Carathéodory’s theorem states that if a point $x \in \mathbb{R}^m$ lies in the convex hull of a set of points $P \subseteq \mathbb{R}^m$, then x is a convex combination of $m+1$ or fewer points of P. There also exists an algorithmic version of Carathéodory’s theorem, shown by Grötschel, Lovász and Schrijver [GLS81]. We will show that the algorithmic version can be applied to the matching polytope, which will enable us to find for arbitrary rational goal points in the matching polytope a convex combination of matchings that results in the goal point.

Theorem 6.3.4 (Carathéodory’s theorem, [Car11]) *Let $k \in \mathbb{N}$ and $P \subseteq \mathbb{R}^k$. For all $x \in \operatorname{conv}(P)$ there are (not necessarily distinct) points $y_0, \ldots, y_k \in P$ such that $x \in \operatorname{conv}(\{y_0, \ldots, y_k\})$.*

Before we state the result of Grötschel, Lovász and Schrijver [GLS81], we adapt some of their terminology.

- A *rational polytope* is a quadruple $(P; n, a_0, T)$, where $P \subseteq \mathbb{R}^n$ is a full-dimensional polytope, $a_0 \in \mathbb{Q}^n$ is a point in the interior of P, and every component of a_0 as well as of every vertex of P is a rational number with numerator and denominator not exceeding T in absolute value.
- A class of rational polytopes $\mathcal{K}$ is called *solvable* if there exists a polynomial-time algorithm that solves the weak separation problem for the members of $\mathcal{K}$. Note that we will assume a fixed encoding for all members of $\mathcal{K}$. The input to the algorithm will hence contain the code of some particular $K \in \mathcal{K}$.

Grötschel, Lovász and Schrijver [GLS81] show the following result.

Theorem 6.3.5 ([GLS81]) *Let $\mathcal{K}$ be a solvable class of rational polytopes. Then there exists an algorithm which, given $(P; n, a_0, T) \in \mathcal{K}$ and a rational vector $y \in P$, yields vertices $x_0, x_1, \ldots, x_n$ of P and coefficients $\lambda_0, \lambda_1, \ldots, \lambda_n \geq 0$ such that $\lambda_0 + \lambda_1 + \cdots + \lambda_n = 1$ and $\lambda_0 x_0 + \lambda_1 x_1 + \cdots + \lambda_n x_n = y$, in time polynomial in n, $\log T$ and $\log S$, where S is the maximum absolute value of numerators and denominators of components of y.*

Corollary 6.3.6 *There is a polynomial-time algorithm that on input of an undirected graph $G = (V, E)$ and a rational vector $x \in M(G)$ computes vertices $y_1, \ldots, y_m \in M(G)$ and a rational $\lambda \in [0,1]^m$ such that $\|\lambda\|_1 = 1$ and $x = \sum_{r=1}^{m} \lambda_r y_r$, where $m = |E| + 1$.*

Proof Consider some input $G = (V, E)$ such that $E \neq \emptyset$ and $x \in \mathbb{Q}^E$ such that $x \in M(G)$. It is easy to see that $a_G = (1/m, 1/m, \ldots, 1/m)^T \in \mathbb{Q}^E$ satisfies each inequality in (6.1) - (6.3) strictly, hence a_G lies in the interior of $M(G)$. The vertices of $M(G)$ are the matchings of G with entries from $\{0, 1\}$ in each component, hence the quadruple $(M(G); |E|, a_G, m)$ is a rational polytope.

Grötschel, Lovász and Schrijver [GLS81] further state that the class of matching polytopes is solvable, which follows from the existence of a polynomial-time algorithm that solves the weak separation problem for $M(G)$ on input G and some vector in $\mathbb{Q}^E$ (cf. Theorem 6.3.2). We can hence apply Theorem 6.3.5.

We run the algorithm from Theorem 6.3.5 on input x and $(M(G); |E|, a_G, m)$, where $M(G)$ is encoded by G, and obtain vertices $x_1, x_2, \ldots, x_m$ of $M(G)$ and coefficients $\lambda_1, \lambda_2, \ldots, \lambda_m \geq 0$ such that $\lambda_1 + \lambda_2 + \cdots + \lambda_m = 1$ and $\lambda_1 x_1 + \lambda_2 x_2 + \cdots + \lambda_m x_m = y$ in time polynomial in m, $\log m$ and $\log S$, where S is the maximum absolute value of numerators and denominators of components of x. Hence the overall running time is polynomial in the length of the input. □

Corollary 6.3.7 *For every $k \geq 1$ there is a polynomial-time algorithm that on input of an $\mathbb{N}^k$-labeled undirected graph $G = (V, E, w)$ and a rational point $x \in M(G)$ computes matchings $M_1, \ldots, M_{k+1}$ in G and $\alpha \in ([0, 1] \cap \mathbb{Q})^{k+1}$, $\|\alpha\|_1 = 1$ such that for all $1 \leq i \leq k$, $w_i(x) = \sum_{r=1}^{k+1} \alpha_r w_i(M_r)$.*

Moreover, if $x \in P(G)$, then $M_1, \ldots, M_{k+1}$ are perfect.

Proof Let $m = |E|+1$. We apply the algorithm from Corollary 6.3.6 to (V, E) and x, and obtain vertices $y_1, \ldots, y_m \in M(G)$ and some rational $\lambda \in [0, 1]^m$ such that $\|\lambda\|_1 = 1$ and $x = \sum_{r=1}^m \lambda_r y_r$. We assume that $\lambda_r > 0$ for all $1 \leq r \leq m$. Recall that each vertex $y_r \in M(G)$ corresponds to a matching M_r of G. Hence we have

$$w(x) = \sum_{e \in E} x_e w(e) = \sum_{e \in E} \sum_{r=1}^m \lambda_r (y_r)_e w(e) = \sum_{r=1}^m \lambda_r \sum_{e \in E} (y_r)_e w(e) = \sum_{r=1}^m \lambda_r w(M_r),$$

so $w(x)$ is contained in the convex hull of the points $w(M_1), \ldots, w(M_m)$.

Moreover, it holds that $w(x), w(M_1), \ldots, w(M_m) \in \mathbb{Q}^k$. By Theorem 6.3.4, there exist $k + 1$ matchings $M_{i_1}, \ldots, M_{i_{k+1}}$ such that $w(x)$ is a convex combination of $w(M_{i_1}), \ldots, w(M_{i_{k+1}})$. Since m is polynomial and k is constant in the length of the input, we can find $M_{i_1}, \ldots, M_{i_{k+1}}$ and the coefficients $\alpha_1, \ldots, \alpha_{k+1}$ by brute force, where in each iteration we only need to solve a system of linear equations of constant size.

Finally, consider the case where $x \in P(G)$ and suppose for some $1 \leq s \leq m$ it holds that M_s is not perfect. Then, $\sum_{e \in E} (y_s)_e < |V|/2$ and hence

$$\begin{aligned}\sum_{e \in E} x_e = \sum_{e \in E} \sum_{r=1}^m \lambda_r (y_r)_e = \sum_{r=1}^m \lambda_r \sum_{e \in E} (y_r)_e = \left(\sum_{r \in \{1, \ldots, m\} - \{s\}} \lambda_r \sum_{e \in E} (y_r)_e \right) + \lambda_s \sum_{e \in E} (y_s)_e \\ \leq (1 - \lambda_s) \frac{|V|}{2} + \lambda_s \sum_{e \in E} (y_s)_e < \frac{|V|}{2},\end{aligned}$$

a contradiction to $x \in P(G)$. Hence $M_1, \ldots, M_m$ and in particular $M_{i_1}, \ldots, M_{i_{k+1}}$ are perfect. □

Convex Combinations of Matchings By Corollary 6.3.7 we already know how to obtain a number of (perfect) matchings from a point in the (perfect) matching polytope, such that the point is a convex combination of the (perfect) matchings. We will now show that from such a set of matchings we can extract a matching that has roughly the same weight as the point in the matching polytope that we started with. Our proof uses the necklace splitting results from the last section.

Theorem 6.3.8 (Convex Combinations of Matchings) *For each $k \geq 1$ there exists a polynomial-time algorithm that on input of an $\mathbb{N}^k$-labeled, directed or undirected graph $G = (V, E, w)$, matchings $M_1, \dots, M_c$ in G with $c \geq 1$, and some $\alpha \in ([0,1] \cap \mathbb{Q})^c$, $\|\alpha\|_1 = 1$, computes a matching M in G such that for each $1 \leq i \leq k$ it holds that*

$$\operatorname{abs}\left(w_i(M) - \sum_{r=1}^{c} \alpha_r w_i(M_r)\right) \leq 16(c-1)k \max_{e \in E} w_i(e).$$

Moreover, if $M_1, \dots, M_c$ are perfect, then M is $8k(c-1)$-near perfect.

Proof The algorithm recurses by decrementing c. Again, the case that $\alpha_c = 1$, which includes the base case $c = 1$, is trivial. Otherwise, we assume that we recursively obtained a ($8k(c-2)$-near perfect) matching M' such that

$$\operatorname{abs}\left(w_i(M') - \sum_{r=1}^{c-1} \frac{\alpha_r}{1-\alpha_c} w_i(M_r)\right) \leq 16(c-2)k \max_{e \in E} w_i(e). \tag{6.7}$$

It remains to combine M' and M_c. For this we first assume that M' and M_c are disjoint perfect matchings. In this case, if we ignore any edge directions, then $M' \cup M_c$ is a set of cycles of even length, which alternate strictly between edges from M' and M_c. Let

$$m_{1,1}, m_{1,2}, m_{2,1}, m_{2,2}, \dots, m_{n,1}, m_{n,2}$$

be an enumeration of the edges in $M' \cup M_c$ such that this sequence alternates strictly between M' and M_c and each cycle (opened at an arbitrary vertex) appears as an infix of this sequence, starting with an edge from M'. Let $\chi\colon \{1, \dots, n\} \to \{1, 2\}$ be the coloring obtained from the application of Corollary 6.2.4 to $v^{j,r} = w(m_{j,r})$ and $(1-\alpha_c, \alpha_c)^T$. This coloring can be found in polynomial time by exhaustive search since it has at most $8k$ jump points and it suffices to find such a coloring that fulfills the error bound. For $M^* = \{m_{j,\chi(j)} \mid j = 1, \dots, n\}$ we obtain for $i = 1, \dots, k$:

$$\operatorname{abs}\big(w_i(M^*) - \big((1-\alpha_c)w_i(M') + \alpha_c w_i(M_c)\big)\big) \leq 8k \max_{e \in E} w_i(e). \tag{6.8}$$

Note that M^* can violate the matching condition but it can be modified to be a matching in G by removing only few edges: Observe that violations of the matching condition can be resolved for each cycle independently, since they do not share vertices. So let $m_{s,1}, \dots, m_{t,2}$ be one of the cycles. Such a violation can only occur if

1. $\chi(j) = 2$ and $\chi(j+1) = 1$ for $s \leq j < t$ or
2. $\chi(s) = 1$ and $\chi(t) = 2$.

Observe that the number of violations is bounded by $8k$, the number of changes of χ, and for each of these violations, it suffices to remove a single edge from M^*. This means that for the matching M obtained by removing these edges from M^*, for $i = 1, \dots, k$ it holds that:

$$\begin{aligned}
&\operatorname{abs}\left(w_i(M) - \sum_{r=1}^{c} \alpha_r w_i(M_r)\right)\\
\leq\ &\operatorname{abs}\left(w_i(M^*) - \sum_{r=1}^{c} \alpha_r w_i(M_r)\right) + 8k \max_{e\in E} w_i(e)\\
\overset{(6.8)}{\leq}\ &\operatorname{abs}\left(\big((1-\alpha_c)w_i(M') + \alpha_c w_i(M_c)\big) - \sum_{r=1}^{c} \alpha_r w_i(M_r)\right) + 16k \max_{e\in E} w_i(e)\\
=\ &\operatorname{abs}\left((1-\alpha_c)w_i(M') - \sum_{r=1}^{c-1} \alpha_r w_i(M_r)\right) + 16k \max_{e\in E} w_i(e)\\
\overset{(6.7)}{\leq}\ &\operatorname{abs}\left((1-\alpha_c)\sum_{r=1}^{c-1} \frac{\alpha_r}{1-\alpha_c}\alpha_r w_i(M_r) - \sum_{r=1}^{c-1} \alpha_r w_i(M_r)\right) + 16k(c-1) \max_{e\in E} w_i(e)\\
=\ &16k(c-1)\max_{e\in E} w_i(e).
\end{aligned}$$

In the case of non-disjoint matchings M' and M_c, we first remove the edges in $M' \cap M_c$ (together with their vertices and all incident edges) from the graph and the matchings, apply the procedure above to the remaining edges and add $M' \cap M_c$ to M afterwards. Observe that the resulting edge set is a matching that fulfills the error bound. If the matchings are not perfect, we add zero-weight edges and possibly a single vertex to the matchings and possibly the graph, apply the procedure and remove them afterwards.

Finally, suppose that M_c is perfect and M' is $8k(c-2)$-near perfect. Then, $|M_c| = {}^{|V|}/_2$ and $|M'| \geq {}^{|V|}/_2 - 8k(c-2)$. Following the above procedure, we first add at most $8k(c-2)$ dummy edges to M' to make it perfect, then combine M' with M_c, make the result a matching by removing at most $8k$ edges, and then remove the dummy edges again to obtain M. Hence $|M| \geq {}^{|V|}/_2 - (8k(c-2) + 8k) = {}^{|V|}/_2 - 8k(c-1)$, so M is $8k(c-1)$-near perfect. □

Deterministic Approximation of Multiobjective Matching Problems We can now develop deterministic approximation algorithms for the problems k-MaxM and k-MaxPM. In the first case we will obtain a PTAS. In the second case, the situation is slightly more complicated, because the matching combination lemma does not always provide perfect matchings. We can, however, show that for every $\varepsilon > 0$ we obtain a polynomial-time algorithm that $(1-\varepsilon)$-approximates every perfect matching by some r-near-perfect matching, where r only depends on the number of objectives. This will be sufficient to obtain a deterministic approximation for k-MaxATSP and k-MaxSTSP, because here we work on complete graphs.

We first show that a gap problem for k-MaxM can be solved in polynomial time.

Lemma 6.3.9 *For every $k \geq 1$ there is a polynomial-time algorithm that on input of an $\mathbb{N}^k$-labeled, undirected graph $G = (V, E, w)$ and $d \in \mathbb{N}^k$ does one of the following:*

- *compute a matching M in G such that $w_i(M) \geq d_i - 16k^2 \max_{e\in E} w_i(e)$ for all $1 \leq i \leq k$*
- *state (correctly) that there is no matching M with $w(M) \geq d$*

Proof We run the algorithm from Corollary 6.3.3 on input G and d and either obtain some $x \in M(G)$ such that $w(x) \geq d$, or the correct assertion that no such x exists. If no such x exists, then there is no matching M of G such that $w(M) \geq d$, and we are done.

Otherwise, let $x \in \mathbb{Q}^E$ with $x \in M(G)$ and $w(x) \geq d$ be the solution we obtained. From Corollary 6.3.7 we get matchings $M_1, \dots, M_{k+1}$ and some $\alpha \in ([0,1] \cap \mathbb{Q})^{k+1}$, $\|\alpha\|_1 = 1$ such that for all $1 \leq i \leq k$ it holds that $w_i(x) = \sum_{r=1}^{k+1} \alpha_r w_i(M_r)$. From Theorem 6.3.8 we obtain a matching M such that for all $1 \leq i \leq k$ it holds that $\operatorname{abs}(w_i(M) - \sum_{r=1}^{k+1} \alpha_r w_i(M_r)) \leq 16k^2 \max_{e \in E} w_i(e)$ and thus $w_i(M) \geq \sum_{r=1}^{k+1} \alpha_r w_i(M_r) - 16k^2 \max_{e\in E} w_i(e) = w_i(x) - 16k^2 \max_{e \in E} w_i(e) \geq d_i - 16k^2 \max_{e \in E} w_i(e)$. □

Note that the error in Lemma 6.3.9 is quite large but still bounded by a constant times the weight of the heaviest edge. We can hence downscale the error by guessing constantly many heavy edges of some optimal matching. This results in the following lemma.

Lemma 6.3.10 *For every $k \geq 1$ and for every $\varepsilon > 0$ there is a polynomial-time algorithm that on input of an $\mathbb{N}^k$-labeled, undirected graph $G = (V, E, w)$ and $d \in \mathbb{N}^k$ does one of the following:*

- *compute a matching M in G such that $w_i(M) \geq (1-\varepsilon)d_i$ for all $1 \leq i \leq k$*
- *state (correctly) that there is no matching M with $w(M) \geq d$*

Proof Let $r = {}^{16k^2}/_{\varepsilon}$. We iterate over all $F \subseteq E$ with $r \leq |F| \leq kr$. For each F we proceed as follows. For each $1 \leq i \leq k$, choose $(\beta_F)_i \in \mathbb{N}$ maximal such that there are distinct edges $e_1, \dots, e_r \in F$ with $w_i(e_j) \geq (\beta_F)_i$ for all $1 \leq j \leq r$. We construct the graph $G_F = (V_F, E_F, w_F)$ from G by the following modifications:

- for each $e \in F$ and each $u \in [e]$ we remove all edges incident to u
- we remove all edges $e \in E$ with $w_j(e) > (\beta_F)_j$ for some $1 \leq j \leq k$

We further define $(d_F)_i = \max(d_i - w_i(F), 0)$. Then we call the algorithm from Lemma 6.3.9 for G_F and $(d_F)_1, \dots, (d_F)_k$. If for some F, the algorithm from Lemma 6.3.9 returns some matching $M_F \subseteq E_F$ of G_F, and $M_F \cup F$ is a matching of G with $w_i(M_F \cup F) \geq (1-\varepsilon)d_i$ for all i, we return $M_F \cup F$. Otherwise, we state that no matching M of G with $w_i(M) \geq d_i$ for all i exists. We will now argue that this algorithm runs in polynomial time and is correct.

Runtime: Since k, ε and hence r are constant, there are polynomially many subsets $F \subseteq E$ of constant size to check. Each check executes the polynomial-time algorithm from Lemma 6.3.9 and performs some polynomial-time operations. So overall the above algorithm runs in polynomial time.

Correctness: Clearly, the case where we return $(M_F \cup F)$ in some iteration is correct. We argue that the other case also is correct.

Suppose we state that no matching M with $w_i(M) \geq d_i$ for all $1 \leq i \leq k$ exists. Then, for each $F \subseteq E$, if the algorithm of Lemma 6.3.9 returns some matching $M_F \subseteq E_F$ of G_F such that $(M_F \cup F)$ is a matching of G, then there exists some $1 \leq j \leq k$ such that $w_j(M_F \cup F) < (1-\varepsilon)d_j$. Now assume that some matching M of G with $w_i(M) \geq d_i$ for all $1 \leq i \leq k$ exists. We show that this leads to a contradiction.

Choose a minimal set $\tilde{F} \subseteq E$ such that for each i, the set $\tilde{F}$ contains the r edges from M with the highest weight in w_i, breaking ties arbitrarily. Then, $r \leq |\tilde{F}| \leq rk$. Consider the iteration with $F = \tilde{F}$. Let $M_F = M - F$ and observe that M_F is a matching in G_F, and it holds that $(w_F)_i(M_F) = w_i(M) - w_i(F) \geq \max(d_i - w_i(F), 0) = (d_F)_i$ for all $1 \leq i \leq k$. Hence the algorithm from Lemma 6.3.9 returns some matching M'_F of G_F such that $(w_F)_i(M'_F) \geq (d_F)_i - 16k^2 \max_{e \in E_F}(w_F)_i(e) \geq (d_F)_i - 16k^2(\beta_F)_i$ for all $1 \leq i \leq k$.

Let $M' = M'_F \cup F$ and observe that M' is a matching in G. Moreover, for each $1 \leq i \leq k$ it holds that

$$\begin{aligned} w_i(M') &= w_i(F) + (w_F)_i(M'_F) \\ &\geq w_i(F) + (d_F)_i - 16k^2(\beta_F)_i \\ &\geq (1-\varepsilon)(w_i(F) + (d_F)_i) + \varepsilon w_i(F) - 16k^2(\beta_F)_i \\ &\geq (1-\varepsilon)d_i + \varepsilon r(\beta_F)_i - 16k^2(\beta_F)_i \\ &= (1-\varepsilon)d_i + \varepsilon\frac{16k^2}{\varepsilon}(\beta_F)_i - 16k^2(\beta_F)_i \\ &= (1-\varepsilon)d_i. \end{aligned}$$

This contradicts the observation that $w_j(M') = w_j(M'_F \cup F) < (1-\varepsilon)d_j$ for some j, hence the assumption was wrong and there exists no matching M of G with $w_i(M) \geq d_i$ for all $1 \leq i \leq k$.

So, in both cases the algorithm works correctly. □

Corollary 6.3.11 *For every $k \geq 1$ there exists a PTAS for k-MaxM.*

Proof Papadimitriou and Yannakakis [PY00] argue that there exists an FPTAS (for arbitrary k-objective problems) if and only if there exists a polynomial-time algorithm that for every input consisting of $b \in \mathbb{N}^k$, problem instance x, and $\varepsilon > 0$, either returns a solution with measure not worse than b in every objective, or correctly answers that no solution exists that is better than d by factor $(1+\varepsilon)$ in each objective. Combined with Lemma 6.3.10, their proof shows Corollary 6.3.11. □

Maximum Weight r-Near Perfect Matching We now state a similar result for r-near perfect matchings.

Lemma 6.3.12 *For every $k \geq 1$ there exists a polynomial-time algorithm that on input of an $\mathbb{N}^k$-labeled undirected graph $G = (V, E, w)$ and $d \in \mathbb{N}^k$ does one of the following:*

- *state (correctly) that there is no perfect matching M with $w(M) \geq d$*
- *compute a $8k^2$-near perfect matching M in G such that $w_i(M) \geq d_i - 16k^2 \max_{e \in E} w_i(e)$ for all i*

Proof If $|V|$ is odd, then there is no perfect matching of G, and we are done. So assume that $|V|$ is even.

We run the algorithm from Corollary 6.3.3 on input G and d and either obtain some $x \in P(G)$ such that $w(x) \geq d$, or the correct assertion that no such x exists. If no such x exists, then there is no perfect matching M of G such that $w(M) \geq d$, and we are done.

Otherwise, let $x \in \mathbb{Q}^E$ with $x \in P(G)$ and $w(x) \geq d$ be the solution we obtained. We apply the algorithm from Corollary 6.3.7 and obtain perfect matchings $M_1, \dots, M_{k+1}$ of G and some $\alpha \in ([0,1] \cap \mathbb{Q})^{k+1}$, $\|\alpha\|_1 = 1$ such that for all $1 \leq i \leq k$ it holds that $d_i \leq w_i(x) = \sum_{r=1}^{k+1} \alpha_r w_i(M_r)$. By the algorithm from Theorem 6.3.8 we obtain an $8k^2$-near perfect matching M of G such that $w_i(M) \geq \sum_{r=1}^{k+1} \alpha_r w_i(M_r) - 16k^2 \max_{e \in E} w_i(e) \geq d_i - 16k^2 \max_{e \in E} w_i(e)$. □

6.4 Deterministic Cycle Cover Approximation

Recall the definition of k-r-MaxDFPCC and k-r-MaxUFCC, which we defined on directed and undirected graphs, respectively, and where we computed (partial) cycle covers that contained at most r fixed edges. We will next show that both problems have good deterministic approximation algorithms. In the case of directed graphs we can find a PTAS. In the case of undirected graphs, we can find an algorithm that works like a PTAS but only returns $8k^2$-near cycle covers.

Cycle Covers in Directed Graphs We first show that k-r-MaxDFPCC admits a PTAS.

Theorem 6.4.1 *For all $k \geq 1$ and $r \geq 0$ there exists a PTAS for k-r-*MaxDFPCC*.*

Proof Let $\varepsilon > 0$. On input of an $\mathbb{N}^k$-labeled, directed graph $G = (V, E, w)$ and $F \subseteq E$ with $|F| \leq r$, we define $G' = (V', E', w')$ by

$$\begin{aligned} V' &= \{1,2\} \times V \\ E' &= \{\{(1,u),(2,v)\} \mid (u,v) \in E \text{ and } u \neq v\} \\ w'_i(e) &= w_i(u,v) && (e = \{(1,u),(2,v)\},\ (u,v) \in E,\ 1 \leq i \leq k) \\ w'_{i+1}(e) &= \begin{cases} 1 & \text{if } (u,v) \in F, \\ 0 & \text{otherwise} \end{cases} && (e = \{(1,u),(2,v)\}, (u,v) \in E). \end{aligned}$$

Observe that every matching M of G' corresponds to a partial cycle cover C of G with the same weight in the first k objectives, and vice versa.

We define $\varepsilon' = \min\{\varepsilon, 1/(r+1)\}$ and call the algorithm from Corollary 6.3.11 for $(k+1)$ and ε' on input G'. We obtain a set of matchings of G', which corresponds to a set of partial cycle covers of G, and return those partial cycle covers that contain F.

The above algorithm runs in time polynomial in $|G| + |F|$. We now argue for the correctness and the approximation ratio of the above algorithm. So let $C \subseteq E$ be a partial cycle cover of G with $F \subseteq C$. Hence there is a matching M of G' such that for all $1 \leq i \leq k$ it holds that $w'_i(M) = w_i(C)$, and $w'_{k+1}(M) = |F|$. The PTAS from Corollary 6.3.11 hence returns a matching M' of G' with $w'_i(M') \geq (1-\varepsilon')w'_i(M)$ for all $1 \leq i \leq k+1$, which corresponds to the partial cycle cover C' of G. So we obtain

$$|F| \geq w'_{k+1}(M') \geq (1 - (\tfrac{1}{r+1})) \cdot |F| = |F| - \tfrac{|F|}{r+1} \geq |F| - \tfrac{r}{r+1} > |F| - 1,$$

hence $w'_{k+1}(M') = |F|$ and thus $F \subseteq C'$. So C' is a partial cycle cover of G returned by the above algorithm. Moreover, C' is an $(1-\varepsilon)$-approximation of C, because for each $1 \leq i \leq k$ it holds that $w_i(C') = w'_i(M') \geq (1-\varepsilon')w'_i(M) \geq (1-\varepsilon)w'_i(M) = (1-\varepsilon)w_i(C)$. □

Cycle Covers in Undirected Graphs The case of undirected graphs is more complicated than the case of directed graphs, so here we obtain a weaker result. Namely, we will show that cycle covers can be approximated arbitrarily good by $8k^2$-near cycle covers. For the purpose of deterministically approximating k-MaxSTSP, this will be sufficient.

Lemma 6.4.2 *For every $k \geq 1$ there exists a polynomial-time algorithm that on input of an $\mathbb{N}^k$-labeled undirected graph $G = (V, E, w)$ and $d_1, \ldots, d_k \in \mathbb{N}$ does one of the following:*

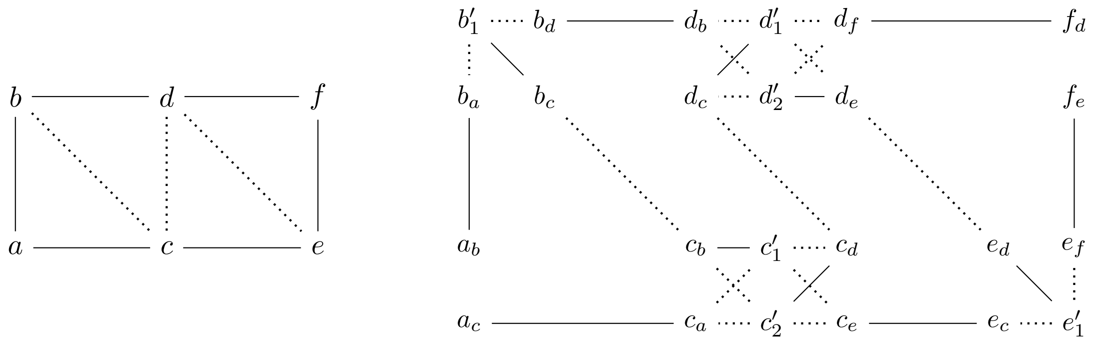
- *state (correctly) that no cycle cover C of G exists such that $w_i(C) \geq d_i$ for all i*
- *output an $8k^2$-near cycle cover C of G such that $w_i(C) \geq d_i - 32k^2 \max_{e \in E} w_i(e)$ for all i*

Proof Recall that $\deg_E(v)$ denotes the number of edges in E incident to v. If there exists some $v \in V$ with $\deg_E(v) < 2$, then there does not exist a cycle cover of G and we are done. So assume that $\deg_E(v) \geq 2$ for all $v \in V$.

We apply Tutte's reduction [Tut54] that reduces the general f-factor problem to the perfect matching problem. Since we can compute $8k^2$-near perfect matchings, we will obtain $8k^2$-near cycle covers. We construct Tutte's [Tut54] reduction graph $\tilde{G} = (V_1 \cup V_2, E_1 \cup E_2, \tilde{w})$ as follows:

$$\begin{aligned}
V_1 &= \{u_v \mid u, v \in V \text{ and } \{u,v\} \in E\} \\
V_2 &= \{v'_i \mid v \in V \text{ and } 1 \leq i \leq (\deg_E(v) - 2)\} \\
E_1 &= \{\{u_v, v_u\} \mid \{u,v\} \in E\} \\
E_2 &= \{\{u_v, u'_i\} \mid u_v \in V_1 \text{ and } u'_i \in V_2\} \\
\tilde{w}(e) &= \begin{cases} w(\{u,v\}) & \text{if } e = \{u_v, v_u\} \in E_1 \\ (0,\dots,0)^T & \text{else} \end{cases}
\end{aligned}$$

Figure 6.1 shows an example of this reduction.



a) Example graph G b) Tutte's reduction graph $\tilde{G}$

Figure 6.1: Example for Tutte's reduction. Suppose we are given the graph G, where the solid edges correspond to a cycle cover. In the reduction graph $\tilde{G}$, this corresponds to a perfect matching (solid edges).

After constructing the graph $\tilde{G}$, we call the algorithm from Lemma 6.3.12 on input $\tilde{G}$ and $d_1, \dots, d_k$. If the algorithm states that no perfect matching $\tilde{M}$ of $\tilde{G}$ exists with $\tilde{w}_i(\tilde{M}) \geq d_i$ for all i, then we state that no cycle cover C of G exists with $w_i(C) \geq d_i$ for all i. Otherwise, let $\tilde{M}$ be the $8k^2$-near perfect matching of $\tilde{G}$ returned by the algorithm from Lemma 6.3.12. For every unmatched $v'_i \in V_2$ we choose an arbitrary $v_u \in V_1$ that is not matched to a vertex in V_2, add $\{v'_i, v_u\}$ to $\tilde{M}$ and remove $\{v_u, u_v\}$ from $\tilde{M}$ (if necessary). Denote the obtained matching by $\tilde{M}'$ and return $C = \{\{u,v\} \in E \mid \{u_v, v_u\} \in \tilde{M}'\}$.

The algorithm described above clearly runs in polynomial time, so it remains to argue for the correctness.

We start with the first case. Observe that every cycle cover C of G can easily be transformed into a perfect matching $\tilde{M}$ of $\tilde{G}$ such that $\tilde{w}(\tilde{M}) = w(C)$. So, if there does not exist a perfect matching $\tilde{M}$ of $\tilde{G}$ such that $\tilde{w}(\tilde{M}) \geq d_i$ for all i, then there does not exist a cycle cover C of G with $w_i(C) \geq d_i$ for all i, hence the first case is correct.

We finish by considering the second case. Since $\tilde{M}$ is $8k^2$-near perfect, we have $2|\tilde{M}| \geq |V_1| + |V_2| - 16k^2$. Since $|\tilde{M}'| \geq |\tilde{M}|$ and in $\tilde{M}'$, every vertex of V_2 is matched, we obtain

$$\begin{aligned} |\{z \in V_1 \mid z \text{ matched in } \tilde{M}'\}| &= 2|\tilde{M}'| - |\{z \in V_2 \mid z \text{ matched in } \tilde{M}'\}| \\ &\geq 2|\tilde{M}| - |V_2| \\ &\geq |V_1| - 16k^2, \end{aligned}$$

hence in $\tilde{M}'$, at most $16k^2$ vertices of V_1 are not matched. Since all vertices of V_2 are matched, for every vertex $v \in V$ there are at most two incident edges in C, hence C is a partial cycle cover of G. Moreover, we need at most $8k^2$ dummy edges of the form $\{u_v, v_u\}$ to make $\tilde{M}'$ perfect, so C is an $8k^2$-near cycle cover of G.

By Lemma 6.3.12, for all $1 \leq i \leq k$ it holds that $\tilde{w}_i(\tilde{M}) \geq d_i - 16k^2 \max_{e \in \tilde{E}} \tilde{w}_i(e) = d_i - 16k^2 \max_{e \in E} w_i(e)$. Furthermore, we removed at most $16k^2$ edges to obtain the matching $\tilde{M}'$, hence we have $w_i(C) = \tilde{w}_i(\tilde{M}') \geq \tilde{w}_i(\tilde{M}) - 16k^2 \max_{e \in \tilde{E}} \tilde{w}_i(e) \geq d_i - 32k^2 \max_{e \in E} w_i(e)$. □

Lemma 6.4.3 *Let $k \geq 1$ and $\varepsilon > 0$. There exists a polynomial time algorithm that on input of an $\mathbb{N}^k$-labeled undirected graph $G = (V, E, w)$ and $d_1, \ldots, d_k \in \mathbb{N}$ does one of the following:*

- *output some $8k^2$-near cycle cover C of G such that $w_i(C) \geq (1 - \varepsilon)d_i$ for all $1 \leq i \leq k$*
- *state (correctly) that no cycle cover C of G exists such that $w_i(C) \geq d_i$ for all $1 \leq i \leq k$*

Proof Let $r = 32k^2/\varepsilon$. We iterate over all $F_1 \subseteq E$ with $r \leq |F_1| \leq kr$ and all $F_2 \subseteq E - F_1$ with $|F_2| \leq 2kr$. Let $F = F_1 \cup F_2$. If F is not a partial cycle cover of G, we continue with the next iteration. Otherwise, for each $1 \leq i \leq k$ we choose $(\beta_F)_i \in \mathbb{N}$ maximal such that there are distinct edges $e_1, \ldots, e_r \in F_1$ with $w_i(e_j) \geq (\beta_F)_i$ for all $1 \leq j \leq r$. Next we construct $G_F = (V, E_F, w_F)$ as follows:

1. $E_F = E - \{\{u, w\} \in (E - F) \mid \exists v \in V \colon \{u, v\} \in F_1\}$
2. for each $e \in F_1$, let $(w_F)_i(e) = 0$ for all $1 \leq i \leq k$
3. for each $e \in E_F - F_1$ with $w_j(e) > (\beta_F)_j$ for some $1 \leq j \leq k$, let $(w_F)_i(e) = 0$ for all $1 \leq i \leq k$
4. for each $e \in E_F - F_1$ with $w_j(e) \leq (\beta_F)_j$ for all $1 \leq j \leq k$, let $w_F(e) = w(e)$

Furthermore, we define $(d_F)_i = \max(d_i - w_i(F_1), 0)$ for all $1 \leq i \leq k$. Next in the iteration, we apply the algorithm from Lemma 6.4.2 to G_F and $(d_F)_1, \ldots, (d_F)_k$. If in some iteration we obtain some $8k^2$-near cycle cover C_F of G_F such that $w_i(C_F \cup F_1) \geq (1 - \varepsilon)d_i$ for all $1 \leq i \leq k$, we return $C_F \cup F_1$. Otherwise we state that no cycle cover C of G exists such that $w_i(C) \geq d_i$ for all $1 \leq i \leq k$.

The algorithm runs in polynomial time, because the size of each set F is bounded by a constant, and hence there are only polynomially many iterations. So it remains to argue for the correctness.

The case where we return the partial cycle cover is correct, because the set $C_F \cup F_1$ is an $8k^2$-near cycle cover of G_F and hence of G, and $C_F \cup F_1$ contains enough weight.

So suppose that we state that no cycle cover C of G with $w_i(C) \geq d_i$ for all $1 \leq i \leq k$ exists. Hence in each iteration defined by a partial cycle cover $F = F_1 \cup F_2 \subseteq E$ of G, if the algorithm from Lemma 6.4.2 returns some $8k^2$-near cycle cover C_F of G_F, then there exists some $1 \leq j \leq k$ such that $w_j(C_F \cup F_1) < (1 - \varepsilon)d_j$. We show that in this case, our statement is also correct.

Assume for a contradiction that there exists some cycle cover C of G with $w_i(C) \geq d_i$ for all $1 \leq i \leq k$. Consider the iteration where $F_1 \subseteq C$ contains the r heaviest edges of C in each objective, breaking ties arbitrarily, and where $F_2 \subseteq C - F_1$ is the set of neighbour edges of F_1 in C. Clearly, $r \leq |F_1| \leq kr$ and $|F_2| \leq 2kr$, so this iteration exists. Moreover, $F = F_1 \cup F_2$ is a partial cycle cover of G. Let $(\beta_F)_1, \dots, (\beta_F)_k$ and $G_F = (V, E_F, w_F)$ as constructed above. Observe that C cannot contain edges in $E - E_F$, hence C is a cycle cover of G_F, and for each $1 \leq i \leq k$ it holds that $(w_F)_i(C) = w_i(C) - w_i(F_1) \geq \max(d_i - w_i(F_1), 0) = (d_F)_i$. Hence the algorithm from Lemma 6.4.2 must return some $8k^2$-near cycle cover $\tilde{C}_F$ of G_F such that $(w_F)_i(\tilde{C}_F) \geq (d_F)_i - 32k^2 \max_{e \in E_F}(w_F)_i(e)$ for all $1 \leq i \leq k$. Moreover, by construction of G_F, the set $\tilde{C}_F \cup F_1$ remains a $8k^2$-near cycle cover of G_F and hence of G. So, for all $1 \leq i \leq k$ we obtain

$$\begin{aligned}
w_i(\tilde{C}_F \cup F_1) &\geq (w_F)_i(\tilde{C}_F) + w_i(F_1) \\
&\geq w_i(F_1) + (d_F)_i - 32k^2 \max_{e \in E_F}(w_F)_i(e) \\
&\geq (1-\varepsilon)(w_i(F_1) + (d_F)_i) + \varepsilon w_i(F_1) - 32k^2(\beta_F)_i \\
&\geq (1-\varepsilon)d_i + \varepsilon w_i(F_1) - 32k^2(\beta_F)_i \\
&\geq (1-\varepsilon)d_i + \varepsilon r(\beta_F)_i - 32k^2(\beta_F)_i \\
&= (1-\varepsilon)d_i + \varepsilon \frac{32k^2}{\varepsilon}(\beta_F)_i - 32k^2(\beta_F)_i \\
&= (1-\varepsilon)d_i,
\end{aligned}$$

a contradiction. Hence the assumption was wrong, and we can correctly state there does not exist a cycle cover C of G with $w_i(C) \geq d_i$ for all $1 \leq i \leq k$. □

Lemma 6.4.4 *For every $k \geq 1$ and $\varepsilon > 0$, there exists a polynomial-time algorithm that on input of an $\mathbb{N}^k$-labeled, undirected graph $G = (V, E, w)$ outputs a set $\mathcal{S}$ of $8k^2$-near cycle covers of G such that for every cycle cover C of G there exists some $C' \in \mathcal{S}$ such that $w_i(C') \geq (1-\varepsilon)w_i(C)$ for all $1 \leq i \leq k$.*

Proof We adapt a proof sketched by Papadimitriou and Yannakakis [PY00].

On input of an undirected, $\mathbb{N}^k$-labeled graph $G = (V, E, w)$, we subdivide the solution value space $\mathbb{N}^k$ into hyperrectangles such that in each dimension, the ratio between the smaller and the bigger value of each hyperrectangle is less than or equal to $(1 - \varepsilon')$ with $\varepsilon' = 1 - \sqrt{1-\varepsilon}$. At each cornerpoint $d_1, \dots, d_k$, we call the algorithm from Lemma 6.4.3 for ε' and G. We initialize $\mathcal{S}$ with the empty set and, for each corner point, if the algorithm from Lemma 6.4.3 returns some solution s, we add s to $\mathcal{S}$. After processing the last cornerpoint, we return $\mathcal{S}$.

Note that there exists a polynomial p such that $w_i(E) \leq 2^{p(|G|)}$ for all $1 \leq i \leq k$, and that $\varepsilon > 0$ is constant. It hence suffices to consider polynomially many cornerpoints. The algorithm from Lemma 6.4.3 is hence called polynomially often, and runs itself in polynomial time, so overall we obtain a polynomial-time algorithm. For the correctness, first recall that the algorithm from Lemma 6.4.3 only returns $8k^2$-near cycle covers of G. Let C be some cycle cover of G. Hence there exists some cornerpoint $d_1, \dots, d_k \in \mathbb{N}$ such that $(1-\varepsilon')w_i(C) < d_i \leq w_i(C)$ for all i. Hence, when called at the point $d_1, \dots, d_k$, the algorithm from Lemma 6.4.3 must return some $8k^2$-near cycle cover C' of G which will then be contained in $\mathcal{S}$, such that $w_i(C') \geq (1-\varepsilon')d_i \geq (1-\varepsilon')^2 w_i(C) = (1-\varepsilon)w_i(C)$ holds for all $1 \leq i \leq k$. □

Lemma 6.4.5 *For every $k \geq 1$, $r \geq 0$ and $\varepsilon > 0$, there exists a polynomial-time algorithm that on input of an $\mathbb{N}^k$-labeled, undirected graph $G = (V, E, w)$ and $F \subseteq E$ with $|F| \leq r$ outputs a set $\mathcal{S}$ of $8k^2$-near cycle covers C' of G with $F \subseteq C'$ such that for every cycle cover C of G with $F \subseteq C$ there exists some $C' \in \mathcal{S}$ such that $w_i(C') \geq (1-\varepsilon)w_i(C)$ for all $1 \leq i \leq k$.*

Proof For all $e \in E$, define w' by $w'_i(e) = w_i(e)$ for all $1 \leq i \leq k$ and $w'_{k+1}(e) = 1$ if $e \in F$ and $w'_{k+1}(e) = 0$ if $e \notin F$. Call the algorithm from Lemma 6.4.4 for $k' = k+1$ and $\varepsilon' = \min(\varepsilon, 1/(r+1))$ on input $G' = (V, E, w')$. Denote the obtained set by $\mathcal{S}$ and return $\mathcal{S}' = \{C \in \mathcal{S} \mid F \subseteq C\}$.

Clearly, $\mathcal{S}'$ is a set of $8k^2$-near cycle covers, each containing F. Suppose that C is a cycle cover of G with $F \subseteq C$. By Lemma 6.4.4 there exists a $8k^2$-near cycle cover $C' \in \mathcal{S}$ of G with $w'_i(C') \geq (1-\varepsilon')w_i(C)$ for all $1 \leq i \leq k+1$. Hence $w_i(C') = w'_i(C') \geq (1-\varepsilon')w_i(C) \geq (1-\varepsilon)w_i(C)$ for all $1 \leq i \leq k$. Moreover, $|F| \geq w'_{k+1}(C') \geq (1-\varepsilon')w'_{k+1}(C) \geq (1-\frac{1}{r+1})|F| = |F| - \frac{|F|}{r+1} > |F| - 1$, hence $w'_{k+1}(C') = |F|$, and $F \subseteq C'$. So C' is contained in $\mathcal{S}'$ and has enough weight to approximate C. □

6.5 Deterministic Approximation of k-MaxTSP

With the deterministic cycle cover approximation algorithms we obtained in the last section we can show how to deterministically approximate k-MaxATSP and k-MaxSTSP. We proceed analogously to the randomized approximation algorithms for k-MaxATSP and k-MaxSTSP that we defined in Section 5.4. On input of some $\mathbb{N}^k$-labeled graph $G = (V, E, w)$ we deterministically compute a $(1-\varepsilon)$-approximate partial cycle cover and use the vector balancing result from the last chapter (Corollary 5.3.2) to remove an edge from each cycle such that in each objective we do not remove too much weight. The remaining paths can be arbitrarily connected to a Hamiltonian cycle. We balance out the error introduced by Corollary 5.3.2 by fixing a constant number of heavy edges. We further note that instead of Corollary 5.3.2 we could use the structured balancing result from this chapter (Corollary 6.2.4). The latter, however, has a bigger error and hence results in higher (but still polynomial) runtime.

Let us first summarize the results of the last section.

Corollary 6.5.1 *For every $k \geq 1$, $r \geq 0$, and $\varepsilon > 0$ there exists a polynomial-time algorithm that on input of a directed (resp., undirected), $\mathbb{N}^k$-labeled graph $G = (V, E, w)$ and $F \subseteq E$ with $|F| \leq r$ computes a set of partial cycle covers $\mathcal{C}$ with the following properties:*

- *$F \subseteq C$ for each $C \in \mathcal{C}$.*
- *For each Hamiltonian cycle R of G with $F \subseteq R$ there exists some $C \in \mathcal{C}$ with $w(C) \geq (1-\varepsilon)w(R)$.*

Proof Suppose we are given a graph $G = (V, E, w)$ and $F \subseteq E$ with $|F| \leq r$.

If G is directed, we call the PTAS from Theorem 6.4.1 and obtain a set $\mathcal{C}$ of partial cycle covers with the following properties:

- $F \subseteq C$ for each $C \in \mathcal{C}$.
- For each partial cycle cover C of G with $F \subseteq C$ there exists some $C' \in \mathcal{C}$ with $w(C') \geq (1-\varepsilon)w(C)$.

Since every Hamiltonian cycle is a partial cycle cover, the assertion holds for directed graphs.

If G is undirected, we call the algorithm from Lemma 6.4.5 and obtain a set $\mathcal{N}$ of $8k^2$-near cycle covers of G with the following properties:

- $F \subseteq N$ for each $N \in \mathcal{N}$.
- For every cycle cover C of G with $F \subseteq C$ there exists some $N \in \mathcal{N}$ with $w(N) \geq (1-\varepsilon)w(C)$.

Since every $8k^2$-near cycle cover and every Hamiltonian cycle is a partial cycle cover, the assertion also holds for undirected graphs. □

We denote the algorithms from Corollary 6.5.1 for directed (resp., undirected), $\mathbb{N}^k$-labeled input graphs by k-$\texttt{MaxDFPCCApprox}_{(r,\varepsilon)}$ (resp., k-$\texttt{MaxUFPCCApprox}_{(r,\varepsilon)}$). Given these algorithms, for every k, c, ε we define the algorithm k-$\texttt{MaxTSPApproxDet}_{(c,\varepsilon)}$ below. The algorithm depends on the parameter c which is used to define the minimum cycle length, and takes directed and undirected complete graphs as inputs. We will show that k-$\texttt{MaxTSPApproxDet}_{(2,\varepsilon)}$, restricted to directed input graphs, deterministically computes a $(1/2 - \varepsilon)$-approximation for k-MaxATSP, and k-$\texttt{MaxTSPApproxDet}_{(3,\varepsilon)}$, restricted to undirected input graphs, deterministically computes a $(2/3 - \varepsilon)$-approximation for k-MaxSTSP.

Algorithm: k-$\texttt{MaxTSPApproxDet}_{(c,\varepsilon)}$ for $k \geq 1$, $c \geq 2$ and $\varepsilon > 0$

```
Input  : $\mathbb{N}^k$-labeled directed/undirected complete graph $G = (V, E, w)$
Output : set of Hamiltonian cycles of $G$
1  let $s = 2ck$ and $r = (c+1)sk$
2  foreach $F = (F_H \cup F_L) \subseteq E$ with $s \leq |F_H| \leq sk$ and $|F_L| \leq c|F_H|$ do
3      let $\beta \in \mathbb{N}^k$ such that $\beta_i = \max\{n \in \mathbb{N} \mid$ there are $s$ edges $e \in F_H$ with $w_i(e) \geq n\}$
4      let $G' = (V, E', w')$, where $E' = F \cup \{e \in E \mid w(e) \leq \beta\}$ and $w' \colon E' \to \mathbb{N}^k$ such that
       $w'(e) = 0$ for all $e \in F_H$ and $w'(e) = w(e)$ for all $e \in (E' - F_H)$
5      call $k$-$\texttt{MaxDFPCCApprox}_{(r,\varepsilon)}(G', F)$ / $k$-$\texttt{MaxUFPCCApprox}_{(r,\varepsilon)}(G', F)$ and obtain a set $\mathcal{C}$
6      foreach partial cycle cover $C \in \mathcal{C}$ of $G$ do
7          let $C_1, \ldots, C_t$ denote the cycles in $C$
8          if for each $i \in \{1, \ldots, t\}$, $(C_i - F_H)$ contains a path of length $c$ then
9              foreach $i \in \{1, \ldots, t\}$ do choose path $e_{i,1}, \ldots, e_{i,c} \in (C_i - F_H)$ arbitrarily
10             compute some coloring $\chi \colon \{1, \ldots, t\} \to \{1, \ldots, c\}$ using the algorithm from
               Corollary 5.3.2 such that $\sum_{i=1}^{t} w(e_{i,\chi(i)}) \leq 2k\beta + \frac{1}{c}\sum_{i=1}^{t}\sum_{j=1}^{c} w(e_{i,j})$
11             remove $\{e_{i,\chi(i)} \mid 1 \leq i \leq t\}$ from $C$ and output the remaining edges in $C$,
               arbitrarily connected with edges from $E$ to a Hamiltonian cycle
```

Theorem 6.5.2 *Let $k \geq 1$ and $\varepsilon > 0$.*

1. *There exists a deterministic $(1/2 - \varepsilon)$-approximation algorithm for k-*MaxATSP.
2. *There exists a deterministic $(2/3 - \varepsilon)$-approximation algorithm for k-*MaxSTSP.

Proof For the problems k-MaxATSP and k-MaxSTSP, we will consider the algorithm k-$\texttt{MaxTSPApproxDet}_{(c,\varepsilon)}$ with $c = 2$ and $c = 3$, restricted to directed and to undirected input graphs, and calling the algorithm k-$\texttt{MaxDFPCCApprox}_{(r,\varepsilon)}$ and k-$\texttt{MaxUFPCCApprox}_{(r,\varepsilon)}$ in line 5, respectively.

In the remainder we will argue for correctness, polynomial runtime, and the correct approximation ratio of the algorithm. So let $G = (V, E, w)$ be some $\mathbb{N}^k$-labeled input graph with $|V|$ large enough.

Correctness: We show that the algorithm is well-defined and outputs Hamiltonian cycles of G. First we iterate over all sets $F = (F_H \cup F_L) \subseteq E$ of bounded cardinality. Fix an arbitrary iteration. We define some β and a subgraph $G' = (V, E', w')$ of G by removing some edges and modifying the weight function. These modifications ensure $w'(e) = w(e) \leq \beta$ for all $e \in (E' - F_H)$ and $w'(e) = 0$ for all $e \in F_H$. Note that $|F| \leq (c+1)sk = r$, hence we can call k-$\texttt{MaxDFPCCApprox}_{(r,\varepsilon)}$ (resp., k-$\texttt{MaxUFPCCApprox}_{(r,\varepsilon)}$) on input (G', F) and obtain a set $\mathcal{C}$ of partial cycle covers of G'. Since $E' \subseteq E$, each $C \in \mathcal{C}$ is a partial cycle cover of G.

Next we iterate over all $C \in \mathcal{C}$. Fix some arbitrary such C and consider all cycles $C_1, \ldots, C_t$ in C. If some cycle $(C_i - F_H)$ does not contain a path of length c, we are done. So suppose this is not the case. Then for each C_i we can choose some path $e_{i,1}, \ldots, e_{i,c}$ arbitrarily. From $e_{i,j} \in C_i \subseteq C \subseteq E'$ and $e_{i,j} \notin F_H$ we obtain that $w(e_{i,j}) \leq \beta$, hence Corollary 5.3.2 applies as required in the algorithm. We obtain a coloring that selects one edge per cycle C_i. We delete each selected edge and end up with a set of paths. Since G is complete, this set can be connected to a Hamiltonian cycle in an arbitrary way. We output this Hamiltonian cycle.

Runtime: First, observe that the cardinality of the set $F = (F_H \cup F_L)$ is bounded by the constant $(c+1)sk$, hence there are only polynomially many such sets, and the outer loop in line 2 is iterated polynomially often. In each iteration, we perform some polynomial-time operations and then call the polynomial-time algorithms from Corollary 6.5.1. Hence the obtained set $\mathcal{C}$ again has polynomial cardinality. This means that the loop in line 6 is iterated polynomially often. Again, in each iteration, some polynomial-time operations are performed. Note that in particular, by Corollary 5.3.2, the coloring in line 10 is computed in polynomial time. So overall we obtain a polynomial-time algorithm.

Approximation Ratio: Let R be some arbitrary Hamiltonian cycle. We show that in some iteration the algorithm outputs a Hamiltonian cycle R' such that $w(R') \geq (1-\varepsilon)w(R)$.

For each $1 \leq i \leq k$, let $F_{H,i} \subseteq R$ be some set of s heaviest edges of R in the i-th objective, 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 possible because we only consider inputs with $|R| = |V|$ large enough. We now have $s \leq |F_H| \leq sk$ and $|F_L| \leq c\,|F_H|$. Hence in line 2 there will be some iteration that chooses F_H and F_L. We fix this iteration for the remainder of the proof.

Let $\beta \in \mathbb{N}^k$ as defined in line 3 and observe that $\beta_i = \min\{w_i(e) \mid e \in F_{H,i}\}$ for all i, which means

$$\beta \leq \frac{1}{s} \cdot w(F_H) \tag{6.9}$$

and that for all edges $e \in (R - F_H)$ we have $w(e) \leq \beta$. Hence it holds that

$$w'(R) = w(R - F_H) = w(R) - w(F_H). \tag{6.10}$$

Next we call the algorithm k-`MaxDFPCCApprox`$_{(r,\varepsilon)}$ (resp., k-`MaxUFPCCApprox`$_{(r,\varepsilon)}$) on input (G', F) and obtain a set of partial cycle covers $\mathcal{C}$ of G. By Corollary 6.5.1 there exists some $C \in \mathcal{C}$ such that $F \subseteq C$ and

$$w'(C) \geq (1-\varepsilon)w'(R). \tag{6.11}$$

Hence in line 6 there will be some iteration that chooses this C. Again we fix this iteration for the remainder of the proof. From $C \subseteq E'$ it follows that $w(e) \leq \beta$ for all $e \in (C - F_H)$, hence

$$w(C - F_H) = w'(C - F_H) = w'(C) - w'(F_H) = w'(C). \tag{6.12}$$

As in line 7, let $C_1, \ldots, C_t$ denote the cycles in C. Note that since C is a partial cycle cover, besides the cycles $C_1, \ldots, C_t$ we also have paths contained in C, and vertices contained in V that are not matched at all. However, a set of edges that only consists of paths can easily be connected to a Hamiltonian cycle without losing weight, so we only have to break up the cycles into paths without losing too much weight. By the choice of c ($c = 2$ for directed graphs, and

$c = 3$ for undirected graphs), 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 line 9 of the algorithm. Note that from $e_{i,j} \in (C - F_H) \subseteq (E' - F_H)$ we obtain $w(e_{i,j}) \leq \beta$ for all i, j.

In line 10 we compute some $\chi\colon \{1, \dots, t\} \to \{1, \dots, c\}$ such that

$$\sum_{i=1}^{t} w(e_{i,\chi(i)}) \leq 2k\beta + \frac{1}{c} \sum_{i=1}^{t} \sum_{j=1}^{c} w(e_{i,j}) \leq 2k\beta + \frac{1}{c} w(C - F_H). \tag{6.13}$$

Recall that by Corollary 5.3.2 such a coloring exists and can be computed in polynomial time. Removing the chosen edges from C breaks all cycles into paths. Hence, after the edge removal, C only contains paths, which can be arbitrarily connected to a Hamiltonian cycle R' with edges from E. We obtain:

$$\begin{aligned}
w(R') &\geq w(C) - \sum_{i=1}^{t} w(e_{i,\chi(i)}) \\
&\geq w(F_H) + w(C - F_H) - 2k\beta - \frac{1}{c} w(C - F_H) && \text{(by (6.13))} \\
&\geq \left(1 - \frac{1}{c}\right) w(C - F_H) + w(F_H) - \frac{2k}{s} w(F_H) && \text{(by (6.9))} \\
&= \left(1 - \frac{1}{c}\right) w'(C) + \left(1 - \frac{2k}{s}\right) w(F_H) && \text{(by (6.12))} \\
&\geq \left(1 - \frac{1}{c}\right) (1 - \varepsilon) w'(R) + \left(1 - \frac{2k}{s}\right) w(F_H) && \text{(by (6.11))} \\
&= \left(1 - \frac{1}{c}\right) (1 - \varepsilon) w(R) + \left(1 - \frac{2k}{s}\right) w(F_H) - \left(1 - \frac{1}{c}\right) (1 - \varepsilon) w(F_H) && \text{(by (6.10))} \\
&= \left(1 - \frac{1}{c} - \varepsilon + \frac{\varepsilon}{c}\right) w(R) + \left(1 - \frac{1}{c}\right) w(F_H) - \left(1 - \frac{1}{c}\right) (1 - \varepsilon) w(F_H) \\
&\geq \left(1 - \frac{1}{c} - \varepsilon\right) w(R)
\end{aligned}$$

So we obtain $w(R') \geq (1/2 - \varepsilon) w(R)$ for directed and $w(R') \geq (2/3 - \varepsilon) w(R)$ for undirected graphs G, respectively. □

6.6 Summary and Discussion

We concluded our study of multiobjective traveling salesman problems by giving deterministic approximation algorithms for its maximization variants. While the randomized results from the last chapter provided a randomized $1/2$ approximation for k-MaxATSP and a randomized $2/3$ approximation for k-MaxSTSP, previously known deterministic algorithms only provided approximation ratios that were linearly dependent on k, the number of objectives (see the work of Manthey [Man12a]). Our results significantly improve this by providing deterministic $(1-\varepsilon)/2$ and $(2-\varepsilon)/3$ approximation algorithms, respectively. Note that our deterministic approximation ratios are arbitrary close to the randomized ones.

In order to obtain the above approximation ratios, we used necklace splitting results that enabled us combine different matchings in a convex way. By looking into the matching polytope, we showed how to obtain deterministic matching algorithms, that in turn gave us cycle cover algorithms by which we could approximate k-MaxATSP and k-MaxSTSP.

We established our results by Corollary 6.2.4, which is very similar to Corollary 5.3.2 from the last chapter. The difference is that Corollary 6.2.4 provides more structure than Corollary 5.3.2, and this is crucial in our application, because it enables us combine (perfect) matchings into a (r-near perfect) matching. It would be interesting to find further applications of this structured balancing result.

Part II

Redundancy of Complete Sets

Chapter 7

Autoreducibility and Mitoticity

In the previous part of this thesis we studied approximation properties of the traveling salesman problem, a very prominent example of an NP-complete problem. In this part, we will shift our focus to properties that generally hold for all complete problems for NP and other classes. Consider, for instance, the following questions:

- Are all complete sets infinite?
- Do all complete sets remain complete if we remove a polynomial-time decidable subset?
- Can all complete sets be split into two equivalent sets?
- Are all complete sets dense?

Some of these questions have already been studied (for instance, Mahaney [Mah82] showed that $\leq_{\mathrm{m}}^{\mathrm{p}}$-complete sets for NP cannot be sparse, unless P = NP), and some questions are answered (for instance, there are no finite $\leq_{\mathrm{m}}^{\mathrm{p}}$-complete sets for EXP, because P $\neq$ EXP). However, for other properties, such basic questions are still open.

Redundancy and Paddability In this part of the thesis we will look at different reducibility notions $\leq$ and complexity classes $\mathcal{C}$ and study the following question:

- How redundant are the $\leq$-complete sets for $\mathcal{C}$?

In the case of $\leq_{\mathrm{m}}^{\mathrm{p}}$-complete sets for NP, this question goes back to the so-called *isomorphism conjecture*, which was raised by Berman and Hartmanis [BH77]. They observed that all *known* $\leq_{\mathrm{m}}^{\mathrm{p}}$-complete problems A for NP are polynomial-time isomorphic to the satisfiability problem SAT (i.e., there exists a bijective reduction function $f \in \mathrm{FP}$ such that $f^{-1} \in \mathrm{FP}$ and $c_A(x) = c_{\mathrm{SAT}}(f(x))$ for all x), and they conjectured that this holds for *all* $\leq_{\mathrm{m}}^{\mathrm{p}}$-complete problems for NP. It turned out that a positive answer to this conjecture implies P $\neq$ NP, because finite sets are not isomorphic to SAT. It is unknown whether the reverse direction holds, and the isomorphism conjecture is still open.

Berman and Hartmanis [BH77] observed that a set is isomorphic to SAT if and only if it is paddable, a property that was easy to show for the known NP-complete problems. They call a set A *paddable* if and only if one can encode and decode in polynomial time arbitrary information into an instance x without changing its membership to A. We can interpret such sets as being very redundant, because the membership information of an arbitrary problem instance x is redundantly stored in the membership information of a large number of different problem instances $y \neq x$. So the isomorphism conjecture is equivalent to the following unsolved redundancy question:

- Are all $\leq_{\mathrm{m}}^{\mathrm{p}}$-complete sets for NP paddable?

Autoreducibility and Mitoticity Since the isomorphism conjecture and hence the paddability question for all $\leq_{\mathrm{m}}^{\mathrm{p}}$-complete sets for NP are still open, it is natural to consider weaker forms of redundancy and ask whether all complete sets for NP at least have these weaker properties. We call a non-trivial set A autoreducible if $c_A(x)$ can be efficiently computed from $c_A(y)$ for $y \neq x$, and we call A mitotic if we can use an efficiently decidable separator to split A into two parts that are equivalent to A and to each other. So we can think of autoreducibility and mitoticity as local and global forms of redundancy, and it is easy to see that these forms of redundancy are weaker than paddability. Figure 7.1 gives an intuition of how autoreducible and mitotic sets are structured. We refer to the next section for a precise definition of these notions.

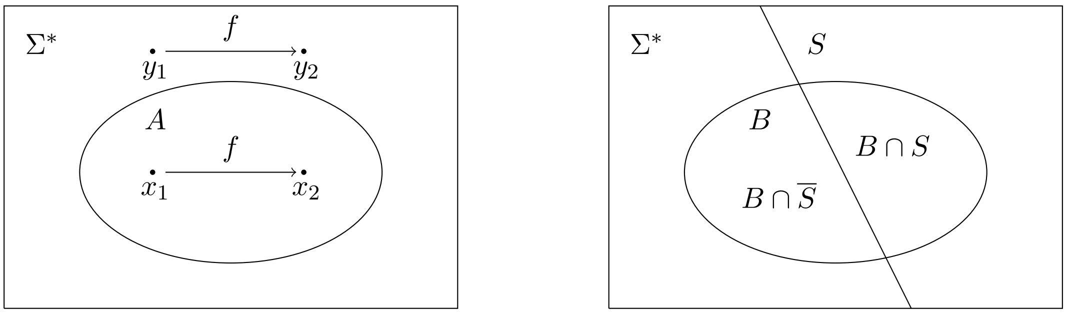


Figure 7.1: Example of autoreducible and mitotic sets over Σ^*. A is $\leq_{\mathrm{m}}^{\mathrm{p}}$-autoreducible via $f \in \mathrm{FP}$, which means that f maps $x_1 \in A$ to some value $x_2 \in A$ with $x_2 \neq x_1$, and f maps $y_1 \notin A$ to some value $y_2 \notin A$ with $y_2 \neq y_1$. B is $\leq_{\mathrm{m}}^{\mathrm{p}}$-mitotic via the separator set $S \in \mathrm{P}$, which means that S splits B into the two parts $B \cap S$ and $B \cap \overline{S}$ such that $B \equiv_{\mathrm{m}}^{\mathrm{p}} B \cap S \equiv_{\mathrm{m}}^{\mathrm{p}} B \cap \overline{S}$.

Glaßer et al. [GOP$^+$07] showed that all non-trivial $\leq_{\mathrm{m}}^{\mathrm{p}}$-complete sets for NP are autoreducible and even mitotic [GPSZ08]. Moreover, a large amount of research has been spent on the study of these properties for complete sets of further complexity classes and with respect to different reducibility notions (see Section 7.2 for a summary of previous and related work). However, most research concentrates on polynomial-time reducibility notions. While we extend this knowledge, our main contribution is a systematic study of autoreducibility and mitoticity properties of complete sets with respect to logspace reducibility notions.

7.1 Definition of Autoreducibility and Mitoticity

Ambos-Spies [AS84] defined autoreducibility, mitoticity, and weak mitoticity as follows.

Definition 7.1.1 (Autoreducibility) *Let A be a set.*

1. *A is called* $\leq_{\mathrm{T}}^{\mathrm{p}}$-autoreducible *if $A \leq_{\mathrm{T}}^{\mathrm{p}} A$ via some polynomial-time oracle Turing machine that on input x never queries x.*
2. *A is called* $\leq_{\mathrm{m}}^{\mathrm{p}}$-autoreducible *if $A \leq_{\mathrm{m}}^{\mathrm{p}} A$ via some $f \in \mathrm{FP}$, where $f(x) \neq x$ for all x.*

We define autoreducibility for the remaining reductions analogously, where the reduction oracle machine or the reduction function satisfies the resource constraints and asks queries or maps to values different from x. For instance, we say that A is $\leq_{\mathrm{dtt}}^{\mathrm{log}}$-autoreducible if $A \leq_{\mathrm{dtt}}^{\mathrm{log}} A$ via some function $f \in \mathrm{FL}$ such that from $f(x) = \langle y_1, \ldots, y_k \rangle$ it follows that $x \notin \{y_1, \ldots, y_k\}$.

Definition 7.1.2 (Mitoticity and Weak Mitoticity) *Let A be a set. A is called* weakly $\leq_{\mathrm{T}}^{\mathrm{p}}$-mitotic *if there exists a set S such that $A \cap S \equiv_{\mathrm{T}}^{\mathrm{p}} A \cap \overline{S} \equiv_{\mathrm{T}}^{\mathrm{p}} A$. If in addition it holds that $S \in \mathrm{P}$, then A is called* $\leq_{\mathrm{T}}^{\mathrm{p}}$-mitotic*. We refer to S as a* separator.

We define mitoticity and weak mitoticity for the remaining reductions analogously. In the case of logspace mitoticity, we require that the separator set is logspace decidable. Moreover, in the case of non-transitive reductions ($\leq^{\mathrm{p}}_{k\text{-tt}}$, for instance) we require that the sets A, $A \cap S$, and $A \cap \overline{S}$ are pairwise equivalent.

7.2 Previous Results and Related Work

Redundancy in General The redundancy concepts we consider have their origins in computability theory. For a set A, Trakhtenbrot [Tra70] defined A to be autoreducible if $A = L(M^A)$ for some oracle Turing machine M that never queries its own input, and Ladner [Lad73] called A mitotic if it is a disjoint union of two sets in the same degree. Ladner showed that for recursively enumerable sets, autoreducibility and mitoticity coincide. Ambos-Spies [AS84] studied the concepts of autoreducibility and mitoticity in the complexity-theoretic setting. He defined the notions of autoreducibility and mitoticity for polynomial-time many-one and Turing reducibility. Moreover, he distinguished between mitoticity and weak mitoticity by also taking the complexity of the separator set into account. While in general, mitoticity implies autoreducibility and weak mitoticity, the converse does not always hold. Ambos-Spies showed that there exists a set that is $\leq^{\mathrm{p}}_{\mathrm{T}}$-autoreducible and not $\leq^{\mathrm{p}}_{\mathrm{T}}$-mitotic. The question whether $\leq^{\mathrm{p}}_{\mathrm{m}}$-mitoticity differs from $\leq^{\mathrm{p}}_{\mathrm{m}}$-autoreducibility remained open until in 2008, Glaßer et al. [GPSZ08] showed that for non-trivial sets, $\leq^{\mathrm{p}}_{\mathrm{m}}$-autoreducibility implies $\leq^{\mathrm{p}}_{\mathrm{m}}$-mitoticity, so here the notions coincide. Their approach also works for $\leq^{\mathrm{p}}_{1\text{-tt}}$-autoreducibility. For space-bounded reducibility notions, Glaßer [Gla10] analyzed a similar approach and showed that for many-one reductions that are allowed to use poly-logarithmic space, autoreducibility implies mitoticity, while an analogous result for logspace many-one reductions does not hold relative to some oracle.

Previous Work on Complete Sets The autoreducibility and mitoticity properties of complete sets are of particular interest in complexity theory. Consequently, there is a considerable amount of research spent on the redundancy of complete sets, which mostly concentrates on polynomial-time complete sets for various complexity classes.

Let $k \geq 2$ and $l \geq 2$. Beigel and Feigenbaum [BF92] showed that for the classes NP, coNP, the levels Σ^{p}_k, Π^{p}_k, and Δ^{p}_k of the polynomial-time hierarchy, and PSPACE, all non-trivial, $\leq^{\mathrm{p}}_{\mathrm{T}}$-complete sets are $\leq^{\mathrm{p}}_{\mathrm{T}}$-autoreducible. For the same classes, Glaßer et al. [GOP⁺07] showed that all non-trivial, $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete sets are $\leq^{\mathrm{p}}_{\mathrm{m}}$-autoreducible, and their result also holds for the reducibility notions $\leq^{\mathrm{p}}_{1\text{-tt}}$, $\leq^{\mathrm{p}}_{\mathrm{dtt}}$, and $\leq^{\mathrm{p}}_{l\text{-dtt}}$. Recall that by Glaßer et al. [GPSZ08], in the case of $\leq^{\mathrm{p}}_{\mathrm{m}}$-autoreducibility and $\leq^{\mathrm{p}}_{1\text{-tt}}$-autoreducibility, we even have $\leq^{\mathrm{p}}_{\mathrm{m}}$-mitoticity and $\leq^{\mathrm{p}}_{1\text{-tt}}$-mitoticity, respectively. These results answered several open questions asked by Buhrman and Torenvliet in a survey paper [BT94] at once. Considering larger complexity classes, Berman [Ber77] showed that all $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete sets for EXP are $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete with respect to length-increasing reductions. This result implies that all $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete sets for EXP are $\leq^{\mathrm{p}}_{\mathrm{m}}$-autoreducible. Moreover, Buhrman et al. [BHT98] showed that every $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete set for EXP is weakly $\leq^{\mathrm{p}}_{\mathrm{m}}$-mitotic. Glaßer et al. [GOP⁺07] improved these results by showing that all $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete sets for EXP are $\leq^{\mathrm{p}}_{\mathrm{m}}$-mitotic. Ganesan and Homer [GH92] showed that all $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete sets for NEXP are $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete with respect to 1-1 reduction functions. Their result implies that all $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete sets A are $\leq^{\mathrm{p}}_{\mathrm{m}}$-autoreducible, because on input x we can look at the reduction function for $0A \cup 1A \leq^{\mathrm{p}}_{\mathrm{m}} A$ on input $0x$ and $1x$. From Glaßer et al. [GPSZ08] we again obtain $\leq^{\mathrm{p}}_{\mathrm{m}}$-mitoticity of these sets. Furthermore, Homer et al. [HKR93] showed that all $\leq^{\mathrm{p}}_{1\text{-tt}}$-complete sets for EXP are $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete, and Buhrman [Buh93] extended this result by showing that all $\leq^{\mathrm{p}}_{1\text{-tt}}$-complete sets for NEXP are $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete. This in particular shows that all $\leq^{\mathrm{p}}_{1\text{-tt}}$-complete sets for EXP and NEXP are

$\le^{\mathrm{p}}_{\mathrm{m}}$-mitotic. We further note that their result also applies in the logspace setting, hence every $\le^{\log}_{1\text{-tt}}$-complete set for PSPACE, EXP and NEXP is $\le^{\log}_{\mathrm{m}}$-complete. Buhrman et al. [BFvMT00] showed that every $\le^{\mathrm{p}}_{\mathrm{T}}$-complete set for EXP is $\le^{\mathrm{p}}_{\mathrm{T}}$-autoreducible, every $\le^{\mathrm{p}}_{\mathrm{tt}}$-complete set for Δ^{p}_{k} is $\le^{\mathrm{p}}_{\mathrm{tt}}$-autoreducible, and every $\le^{\mathrm{p}}_{2\text{-tt}}$-complete set for EXP is $\le^{\mathrm{p}}_{2\text{-tt}}$-autoreducible. Moreover, there exists a set that is $\le^{\mathrm{p}}_{3\text{-tt}}$-complete for EXP and not $\le^{\mathrm{p}}_{\mathrm{btt}}$-autoreducible [BFvMT00], and a set that is $\le^{\mathrm{p}}_{\mathrm{T}}$-complete for NEXP and not $\le^{\mathrm{p}}_{\mathrm{tt}}$-autoreducible [NS14].

7.3 Contributions in this Part

Polynomial-Time Bounded Reducibility Notions Table 7.1 summarizes the known autoreducibility and mitoticity results in the polynomial-time setting.

$\le$	NP	coNP	Σ^{p}_{k}	Π^{p}_{k}	Δ^{p}_{k}	PSPACE	EXP	NEXP
$\le^{\mathrm{p}}_{\mathrm{m}}$	$\mathrm{M}^{\mathrm{p}}_{\mathrm{m}}$	$\mathrm{M}^{\mathrm{p}}_{\mathrm{m}}$	$\mathrm{M}^{\mathrm{p}}_{\mathrm{m}}$	$\mathrm{M}^{\mathrm{p}}_{\mathrm{m}}$	$\mathrm{M}^{\mathrm{p}}_{\mathrm{m}}$	$\mathrm{M}^{\mathrm{p}}_{\mathrm{m}}$	$\mathrm{M}^{\mathrm{p}}_{\mathrm{m}}$	$\mathrm{M}^{\mathrm{p}}_{\mathrm{m}}$
$\le^{\mathrm{p}}_{1\text{-tt}}$	$\mathrm{M}^{\mathrm{p}}_{1\text{-tt}}$	$\mathrm{M}^{\mathrm{p}}_{1\text{-tt}}$	$\mathrm{M}^{\mathrm{p}}_{1\text{-tt}}$	$\mathrm{M}^{\mathrm{p}}_{1\text{-tt}}$	$\mathrm{M}^{\mathrm{p}}_{1\text{-tt}}$	$\mathrm{M}^{\mathrm{p}}_{1\text{-tt}}$	$\mathrm{M}^{\mathrm{p}}_{\mathrm{m}}$	$\mathrm{M}^{\mathrm{p}}_{\mathrm{m}}$
$\le^{\mathrm{p}}_{l\text{-dtt}}$	$\mathrm{A}^{\mathrm{p}}_{l\text{-dtt}}$	$\mathrm{A}^{\mathrm{p}}_{l\text{-dtt}}$	$\mathrm{A}^{\mathrm{p}}_{l\text{-dtt}}$	$\mathrm{A}^{\mathrm{p}}_{l\text{-dtt}}$	$\mathrm{A}^{\mathrm{p}}_{l\text{-dtt}}$	$\mathrm{W}^{\mathrm{p}}_{s\text{-dtt}}$, $\mathrm{A}^{\mathrm{p}}_{l\text{-dtt}}$	$\mathrm{M}^{\mathrm{p}}_{l\text{-dtt}}$	$\mathrm{A}^{\mathrm{p}}_{l\text{-dtt}}$
$\le^{\mathrm{p}}_{\mathrm{dtt}}$	$\mathrm{A}^{\mathrm{p}}_{\mathrm{dtt}}$	$\mathrm{A}^{\mathrm{p}}_{\mathrm{dtt}}$	$\mathrm{A}^{\mathrm{p}}_{\mathrm{dtt}}$	$\mathrm{A}^{\mathrm{p}}_{\mathrm{dtt}}$	$\mathrm{A}^{\mathrm{p}}_{\mathrm{dtt}}$	$\mathrm{W}^{\mathrm{p}}_{\mathrm{dtt}}$, $\mathrm{A}^{\mathrm{p}}_{\mathrm{dtt}}$	$\mathrm{M}^{\mathrm{p}}_{\mathrm{dtt}}$	$\mathrm{A}^{\mathrm{p}}_{\mathrm{dtt}}$
$\le^{\mathrm{p}}_{l\text{-ctt}}$					$\mathrm{A}^{\mathrm{p}}_{l\text{-tt}}$	$\mathrm{A}^{\mathrm{p}}_{l\text{-tt}}$	$\mathrm{M}^{\mathrm{p}}_{l\text{-ctt}}$	$\mathrm{A}^{\mathrm{p}}_{l\text{-ctt}}$
$\le^{\mathrm{p}}_{\mathrm{ctt}}$						$\mathrm{A}^{\mathrm{p}}_{\mathrm{tt}}$	$\mathrm{M}^{\mathrm{p}}_{\mathrm{ctt}}$	$\mathrm{A}^{\mathrm{p}}_{\mathrm{ctt}}$
$\le^{\mathrm{p}}_{2\text{-tt}}$							$\mathrm{M}^{\mathrm{p}}_{2\text{-tt}}$	$\mathrm{A}^{\mathrm{p}}_{2\text{-tt}}$
$\le^{\mathrm{p}}_{\mathrm{btt}}$							$\not\mathrm{A}^{\mathrm{p}}_{\mathrm{btt}}$	
$\le^{\mathrm{p}}_{\mathrm{tt}}$					$\mathrm{A}^{\mathrm{p}}_{\mathrm{tt}}$			
$\le^{\mathrm{p}}_{\mathrm{T}}$	$\mathrm{A}^{\mathrm{p}}_{\mathrm{T}}$	$\mathrm{A}^{\mathrm{p}}_{\mathrm{T}}$	$\mathrm{A}^{\mathrm{p}}_{\mathrm{T}}$	$\mathrm{A}^{\mathrm{p}}_{\mathrm{T}}$	$\mathrm{A}^{\mathrm{p}}_{\mathrm{T}}$	$\mathrm{A}^{\mathrm{p}}_{\mathrm{T}}$	$\mathrm{A}^{\mathrm{p}}_{\mathrm{T}}$	$\not\mathrm{A}^{\mathrm{p}}_{\mathrm{tt}}$

Table 7.1: Overview of polynomial-time redundancy results of non-trivial complete sets. It holds that $k \ge 2$, $l \ge 2$, and $s = l^3 + l^2 + l$. In row $\le$ and column $\mathcal{C}$, the entry $\mathrm{A}^{\mathrm{p}}_{t}$ means that every $\le$-complete set for $\mathcal{C}$ is $\le^{\mathrm{p}}_{t}$-autoreducible, the entry $\mathrm{M}^{\mathrm{p}}_{t}$ means that every $\le$-complete set for $\mathcal{C}$ is $\le^{\mathrm{p}}_{t}$-mitotic, and the entry $\mathrm{W}^{\mathrm{p}}_{t}$ means that every $\le$-complete set for $\mathcal{C}$ is weakly $\le^{\mathrm{p}}_{t}$-mitotic. The prefix $\not\,$ negates the statement, i.e., there are complete sets that are not autoreducible. Entries that follow from universal relations between reducibility notions are omitted (for instance, all $\le^{\mathrm{p}}_{\mathrm{ctt}}$-complete sets for NP are $\le^{\mathrm{p}}_{\mathrm{T}}$-complete and hence $\le^{\mathrm{p}}_{\mathrm{T}}$-autoreducible).

In the polynomial-time setting, most of our contributions are on large complexity classes such as EXP and NEXP. Here we can define complete sets that are strong enough to simulate and hence diagonalize against polynomial-time reducibilities. We apply this technique in Chapter 10 and obtain that for EXP, all sets that are complete with respect to the reducibility notions $\le^{\mathrm{p}}_{k\text{-ctt}}$, $\le^{\mathrm{p}}_{\mathrm{ctt}}$, $\le^{\mathrm{p}}_{k\text{-dtt}}$, $\le^{\mathrm{p}}_{\mathrm{dtt}}$, and $\le^{\mathrm{p}}_{2\text{-tt}}$ are mitotic, and for NEXP, all complete sets for the reducibility notions $\le^{\mathrm{p}}_{k\text{-ctt}}$, $\le^{\mathrm{p}}_{\mathrm{ctt}}$, $\le^{\mathrm{p}}_{k\text{-dtt}}$, $\le^{\mathrm{p}}_{\mathrm{dtt}}$, and $\le^{\mathrm{p}}_{2\text{-tt}}$ are autoreducible. For weaker classes, the diagonalization technique does not apply, and we need other approaches. In Chapter 11, we apply the deterministic coin tossing approach to show how to turn autoreducibility into weak mitoticity. We obtain that all $\le^{\mathrm{p}}_{l\text{-dtt}}$-complete sets for PSPACE are weakly $\le^{\mathrm{p}}_{s\text{-dtt}}$-mitotic, where $l \ge 2$ and $s = l^3 + l^2 + l$, and all $\le^{\mathrm{p}}_{\mathrm{dtt}}$-complete sets for PSPACE are weakly $\le^{\mathrm{p}}_{\mathrm{dtt}}$-mitotic.

Logspace Bounded Reducibility Notions Table 7.2 summarizes the logspace results that we show in this thesis.

$\leq$	NL	P	$\Sigma_k^{\mathrm{p}}, \Pi_k^{\mathrm{p}}$	$\Delta_{k+1}^{\mathrm{p}}$	PSPACE	EXP	NEXP
$\leq_{\mathrm{m}}^{\log}$	$\mathrm{A}_{1\text{-tt}}^{\log}$, $\mathrm{A}_{2\text{-dtt}}^{\log}$, $\mathrm{A}_{2\text{-ctt}}^{\log}$	$\mathrm{A}_{1\text{-tt}}^{\log}$, $\mathrm{W}_{2\text{-tt}}^{\log}$	$\mathrm{A}_{\mathrm{T}}^{\log}$	$\mathrm{A}_{1\text{-tt}}^{\log}$, $\mathrm{W}_{2\text{-tt}}^{\log}$	$\mathrm{M}_{\mathrm{m}}^{\log}$	$\mathrm{M}_{\mathrm{m}}^{\log}$	$\mathrm{A}_{\mathrm{m}}^{\log}$, $\mathrm{W}_{2\text{-dtt}}^{\log}$
$\leq_{l\text{-dtt}}^{\log}$	$\mathrm{A}_{l\text{-tt}}^{\log}$, $\mathrm{A}_{2l\text{-dtt}}^{\log}$	$\mathrm{A}_{l\text{-tt}}^{\log}$	$\mathrm{A}_{\mathrm{T}}^{\log}$	$\mathrm{A}_{l\text{-tt}}^{\log}$	$\mathrm{M}_{l\text{-dtt}}^{\log}$	$\mathrm{M}_{l\text{-dtt}}^{\log}$	$\mathrm{A}_{l\text{-dtt}}^{\log}$
$\leq_{\mathrm{dtt}}^{\log}$	$\mathrm{A}_{\mathrm{dtt}}^{\log}$	$\mathrm{A}_{\mathrm{tt}}^{\log}$	$\mathrm{A}_{\mathrm{T}}^{\log}$	$\mathrm{A}_{\mathrm{tt}}^{\log}$	$\mathrm{M}_{\mathrm{dtt}}^{\log}$	$\mathrm{M}_{\mathrm{dtt}}^{\log}$	$\mathrm{A}_{\mathrm{dtt}}^{\log}$
$\leq_{l\text{-ctt}}^{\log}$	$\mathrm{A}_{l\text{-tt}}^{\log}$, $\mathrm{A}_{2l\text{-ctt}}^{\log}$	$\mathrm{A}_{l\text{-tt}}^{\log}$		$\mathrm{A}_{l\text{-tt}}^{\log}$	$\mathrm{M}_{l\text{-ctt}}^{\log}$	$\mathrm{M}_{l\text{-ctt}}^{\log}$	$\mathrm{A}_{l\text{-ctt}}^{\log}$
$\leq_{\mathrm{ctt}}^{\log}$	$\mathrm{A}_{\mathrm{ctt}}^{\log}$	$\mathrm{A}_{\mathrm{tt}}^{\log}$		$\mathrm{A}_{\mathrm{tt}}^{\log}$	$\mathrm{M}_{\mathrm{ctt}}^{\log}$	$\mathrm{M}_{\mathrm{ctt}}^{\log}$	$\mathrm{A}_{\mathrm{ctt}}^{\log}$
$\leq_{1\text{-tt}}^{\log}$	$\mathrm{A}_{2\text{-tt}}^{\log}$	$\mathrm{A}_{2\text{-tt}}^{\log}$	$\mathrm{A}_{\mathrm{T}}^{\log}$	$\mathrm{A}_{\log\text{-T}}^{\log[1]}$	$\mathrm{M}_{\mathrm{m}}^{\log}$	$\mathrm{M}_{\mathrm{m}}^{\log}$	$\mathrm{A}_{\mathrm{m}}^{\log}$, $\mathrm{W}_{2\text{-dtt}}^{\log}$
$\leq_{2\text{-tt}}^{\log}$	$\mathrm{A}_{\log\text{-T}}^{\log[1]}$	$\mathrm{A}_{\log\text{-T}}^{\log[1]}$		$\mathrm{A}_{\log\text{-T}}^{\log[1]}$	$\mathrm{M}_{2\text{-tt}}^{\log}$	$\mathrm{M}_{2\text{-tt}}^{\log}$	$\mathrm{A}_{2\text{-tt}}^{\log}$
$\leq_{\mathrm{btt}}^{\log}$	$\mathrm{A}_{\log\text{-T}}^{\log[1]}$	$\mathrm{A}_{\log\text{-T}}^{\log[1]}$		$\mathrm{A}_{\log\text{-T}}^{\log[1]}$	$\not\forall\mathrm{A}_{\mathrm{btt}}^{\log}$		
$\leq_{\mathrm{tt}}^{\log}$	$\mathrm{A}_{\mathrm{tt}}^{\log}$	$\mathrm{A}_{\mathrm{tt}}^{\log}$		$\mathrm{A}_{\mathrm{tt}}^{\log}$			

Table 7.2: Summary of the redundancy results of non-trivial logspace complete sets we show in this thesis. It holds that $k \geq 1$ and $l \geq 2$. In row $\leq$ and column $\mathcal{C}$, the entry $\mathrm{A}_t^{\log}$ means that every $\leq$-complete set for $\mathcal{C}$ is $\leq_t^{\log}$-autoreducible, the entry $\mathrm{M}_t^{\log}$ means that every $\leq$-complete set for $\mathcal{C}$ is $\leq_t^{\log}$-mitotic, and the entry $\mathrm{W}_t^{\log}$ means that every $\leq$-complete set for $\mathcal{C}$ is weakly $\leq_t^{\log}$-mitotic. The prefix $\not\forall$ negates the statement, i.e., there are complete sets that are not autoreducible. Note that $\mathrm{NP} = \Sigma_1^{\mathrm{p}}$ and $\mathrm{coNP} = \Pi_1^{\mathrm{p}}$.

In the logspace setting, we systematically study the autoreducibility and mitoticity properties of complete sets for different reducibility notions and complexity classes. We apply different techniques, depending on the general computational strength of the class and on the reducibility notion we consider. For small classes such as NL and P, a logspace computation can follow a single computation path in the configuration graph of an appropriate machine model (nondeterministic and alternating logspace Turing machines). In Chapter 8 we use self-reducibility of complete sets to generalize this property, which leads to general autoreducibility results. For classes that are sufficiently large such as PSPACE, EXP, and NEXP, the diagonalization technique in Chapter 10 shows that complete sets are autoreducible or even mitotic. In Chapter 9 we consider intermediate classes such as the different levels of the polynomial-time hierarchy. Local checkability of transcripts of computations and similar objects shows autoreducibility results for complete sets with respect to reducibility notions such as $\leq_{\mathrm{btt}}^{\log}$ and $\leq_{\mathrm{tt}}^{\log}$. Moreover, we show how to use this technique to translate autoreducibility results from the polynomial-time setting to the logspace setting, which results in logspace autoreducibility results for complete sets of classes such as NP and coNP. In Chapter 11 we finish our study by showing that in some cases, logspace autoreducibility implies weak logspace mitoticity.

7.4 Simple Properties

We finish this chapter by providing some general properties about autoreducibility and mitoticity, and complete sets. In particular, we obtain that for classes that are closed under complementation, it is easy to show autoreducibility of complete sets (with respect to a weaker reducibility notion).

For non-trivial sets, it is easy to show that mitoticity implies weak mitoticity and autoreducibility.

Proposition 7.4.1 *Let A be a non-trivial set.*

1. *If A is $\leq_{\mathrm{m}}^{\mathrm{p}}$-mitotic, then A is weakly $\leq_{\mathrm{m}}^{\mathrm{p}}$-mitotic.*
2. *If A is $\leq_{\mathrm{m}}^{\mathrm{p}}$-mitotic, then A is $\leq_{\mathrm{m}}^{\mathrm{p}}$-autoreducible.*

Proof Let A be $\leq_{\mathrm{m}}^{\mathrm{p}}$-mitotic. From the definition of mitoticity and weak mitoticity we directly obtain that A is weakly $\leq_{\mathrm{m}}^{\mathrm{p}}$-mitotic. Moreover, since A is $\leq_{\mathrm{m}}^{\mathrm{p}}$-mitotic, there exists a separator $S \in \mathrm{P}$ such that $A \cap S \leq_{\mathrm{m}}^{\mathrm{p}} A \cap \overline{S}$ via $f \in \mathrm{FP}$ and $A \cap \overline{S} \leq_{\mathrm{m}}^{\mathrm{p}} A \cap S$ via $g \in \mathrm{FP}$.

On input x, we first test whether $x \in S$ holds or not. Suppose $x \in S$ (the other case works analogously) and recall that $c_{A\cap S}(x) = c_{A\cap\overline{S}}(f(x))$. Let $y := f(x)$. If $y \in S$, then $c_{A\cap\overline{S}}(y) = 0$, hence $x \notin A$, and so we can return a fixed value $x_0 \neq x$ with $x_0 \notin A$. If $y \in \overline{S}$, then $y \neq x$, and we have $c_A(x) = c_{A\cap S}(x) = c_{A\cap\overline{S}}(y) = c_A(y)$. This shows that A is $\leq_{\mathrm{m}}^{\mathrm{p}}$-autoreducible. □

It is not difficult to see that the above proposition generally holds for the reducibility notions we consider in this thesis. In general, the converse directions of the above implications are more interesting. For instance, Ambos-Spies [AS84] showed that $\leq_{\mathrm{T}}^{\mathrm{p}}$-autoreducibility does not imply $\leq_{\mathrm{T}}^{\mathrm{p}}$-mitoticity, while Glaßer et al. [GPSZ08] showed that $\leq_{\mathrm{m}}^{\mathrm{p}}$-autoreducibility implies $\leq_{\mathrm{m}}^{\mathrm{p}}$-mitoticity for non-trivial sets. So in general, we can think of mitoticity as a stronger form of redundancy than autoreducibility and weak mitoticity, which for some reducibility notions coincide.

We further have simple observations regarding autoreducibility of sets and their complements. We first need some properties of complexity classes and complete sets.

Proposition 7.4.2 *Let A be a set, let $\mathcal{C}$ be some complexity class, and for $k \geq 1$ and $r \in \{\mathrm{p}, \log\}$, let $\leq$ be one of the following reducibility notions: $\leq_{\mathrm{m}}^{r}$, $\leq_{k\text{-tt}}^{r}$, $\leq_{k\text{-T}}^{r}$, $\leq_{\mathrm{btt}}^{r}$, $\leq_{\mathrm{tt}}^{r}$, $\leq_{\log\text{-T}}^{r}$, $\leq_{\mathrm{T}}^{r}$.*

1. *If A is $\leq$-complete for $\mathcal{C}$, then $\overline{A}$ is $\leq$-complete for $\mathrm{co}\mathcal{C}$.*
2. *If A is $\leq_{k\text{-ctt}}^{r}$-complete for $\mathcal{C}$, then $\overline{A}$ is $\leq_{k\text{-dtt}}^{r}$-complete for $\mathrm{co}\mathcal{C}$.*
3. *If A is $\leq_{\mathrm{ctt}}^{r}$-complete for $\mathcal{C}$, then $\overline{A}$ is $\leq_{\mathrm{dtt}}^{r}$-complete for $\mathrm{co}\mathcal{C}$.*
4. *If A is $\leq_{k\text{-dtt}}^{r}$-complete for $\mathcal{C}$, then $\overline{A}$ is $\leq_{k\text{-ctt}}^{r}$-complete for $\mathrm{co}\mathcal{C}$.*
5. *If A is $\leq_{\mathrm{dtt}}^{r}$-complete for $\mathcal{C}$, then $\overline{A}$ is $\leq_{\mathrm{ctt}}^{r}$-complete for $\mathrm{co}\mathcal{C}$.*

Proof Let $B \in \mathrm{co}\mathcal{C}$, hence $\overline{B} \in \mathcal{C}$, and $\overline{B}$ is reducible to A.

In the case of $\leq_{k\text{-tt}}^{r}, \leq_{k\text{-T}}^{r}, \leq_{\mathrm{btt}}^{r}, \leq_{\mathrm{tt}}^{r}, \leq_{\log\text{-T}}^{r}, \leq_{\mathrm{T}}^{r}$ we can simulate the reduction from $\overline{B}$ to A with queries to $\overline{A}$, where we invert each answer and accept if and only the simulation rejects. In the case of $\leq_{\mathrm{m}}^{r}$ we can simply use the same reduction function. Hence $B \leq \overline{A}$, and item 1 holds.

If A is $\leq_{k\text{-ctt}}^{r}$-complete for $\mathcal{C}$, then there exists a reduction function f in FP or even in FL from $\overline{B}$ to A such that $c_{\overline{B}}(x) = \min\{c_A(y) \mid y \in f(x)\}$ and $c_B(x) = 1 - \min\{c_A(y) \mid y \in f(x)\} = \max\{c_{\overline{A}}(y) \mid y \in f(x)\}$ for all x. The case where the number of queries is not bounded by a constant is shown analogously. Hence item 2 and item 3 hold.

Item 4 and item 5 are shown analogously, where we replace max with min and vice versa. □

Proposition 7.4.3 *Let $\mathcal{C}$ be a complexity class that is closed under complementation. For every Boolean function $\alpha\colon \{0,1\}^k \to \{0,1\}$, if A is $\leq_{\alpha\mathrm{tt}}^{\log}$-complete for $\mathcal{C}$, then A is $\leq_{\overline{\alpha}\mathrm{tt}}^{\log}$-complete for $\mathcal{C}$, where $\overline{\alpha}\colon \{0,1\}^k \to \{0,1\}$ with $\overline{\alpha}(b_1, \ldots, b_k) = 1 - \alpha(b_1, \ldots, b_k)$ for all $b_1, \ldots, b_k \in \{0,1\}$.*

Proof Let $B \in \mathcal{C}$. Since $\overline{B} \in \mathcal{C}$, there is a function f that shows $\overline{B} \leq^{\log}_{\alpha\text{tt}} A$. On input x, consider $\langle y_1, y_2, \ldots, y_k \rangle := f(x)$. We now have $c_B(x) = 1 - c_{\overline{B}}(x) = 1 - \alpha(c_A(y_1), c_A(y_2), \ldots, c_A(y_k)) = \overline{\alpha}(c_A(y_1), c_A(y_2), \ldots, c_A(y_k))$, and thus f also shows $B \leq^{\log}_{\overline{\alpha}\text{tt}} A$. □

Proposition 7.4.4 *Let A be a non-trivial set, and for $k \geq 1$ and $r \in \{\text{p}, \log\}$, let $\leq$ be one of the following reducibility notions: $\leq^r_{\text{m}}, \leq^r_{k\text{-tt}}, \leq^r_{k\text{-T}}, \leq^r_{\text{btt}}, \leq^r_{\text{tt}}, \leq^r_{\text{log-T}}, \leq^r_{\text{T}}$.*

1. *If A is $\leq$-autoreducible, then $\overline{A}$ is $\leq$-autoreducible.*
2. *If A is $\leq^r_{k\text{-ctt}}$-autoreducible, then $\overline{A}$ is $\leq^r_{k\text{-dtt}}$-autoreducible.*
3. *If A is $\leq^r_{\text{ctt}}$-autoreducible, then $\overline{A}$ is $\leq^r_{\text{dtt}}$-autoreducible.*
4. *If A is $\leq^r_{k\text{-dtt}}$-autoreducible, then $\overline{A}$ is $\leq^r_{k\text{-ctt}}$-autoreducible.*
5. *If A is $\leq^r_{\text{dtt}}$-autoreducible, then $\overline{A}$ is $\leq^r_{\text{ctt}}$-autoreducible.*

Proof In the case of $\leq^r_{k\text{-tt}}, \leq^r_{k\text{-T}}, \leq^r_{\text{btt}}, \leq^r_{\text{tt}}, \leq^r_{\text{log-T}}, \leq^r_{\text{T}}$ we can simulate the autoreduction for A with queries to $\overline{A}$, where we invert each answer and accept if and only the simulation rejects. In the case of $\leq^r_{\text{m}}$ we can simply use the same autoreduction function. This shows item 1.

If A is $\leq^r_{k\text{-ctt}}$-autoreducible, then there exists an autoreduction function f in FP or even in FL that maps x to k queries $y_1, \ldots, y_k$ different from x such that $c_A(x) = \min\{c_A(y) \mid y \in f(x)\}$ and hence $c_{\overline{A}}(x) = 1 - c_A(x) = 1 - \min\{c_A(y) \mid y \in f(x)\} = \max\{c_{\overline{A}}(y) \mid y \in f(x)\}$ for all x. The case where the number of queries is not bounded by a constant is shown analogously. Hence item 2 and item 3 hold.

Item 4 and item 5 are shown analogously, where we replace max by min and vice versa. □

Proposition 7.4.5 *Let $k \geq 1$, $r \in \{\text{p}, \log\}$, let A be a non-trivial set, and $\mathcal{C}$ be closed under complementation.*

1. *If A is $\leq^r_{\text{m}}$-complete for $\mathcal{C}$, then A is $\leq^r_{1\text{-tt}}$-autoreducible.*
2. *If A is $\leq^r_{k\text{-dtt}}$-complete for $\mathcal{C}$, then A is $\leq^r_{k\text{-tt}}$-autoreducible.*
3. *If A is $\leq^r_{k\text{-ctt}}$-complete for $\mathcal{C}$, then A is $\leq^r_{k\text{-tt}}$-autoreducible.*
4. *If A is $\leq^r_{\text{dtt}}$-complete for $\mathcal{C}$, then A is $\leq^r_{\text{tt}}$-autoreducible.*
5. *If A is $\leq^r_{\text{ctt}}$-complete for $\mathcal{C}$, then A is $\leq^r_{\text{tt}}$-autoreducible.*

Proof Let A be $\leq^r_{k\text{-dtt}}$-complete for $\mathcal{C}$. Since $\mathcal{C}$ is closed under complementation, we have $\overline{A} \in \mathcal{C}$ and thus $\overline{A} \leq^r_{k\text{-dtt}} A$ via some f in FP or even in FL. This means $c_A(x) = 1 - \max\{c_A(y) \mid y \in f(x)\}$, where $f(x) = \{y_1, \ldots, y_k\}$. If $x \notin f(x)$, this yields a $\leq^r_{k\text{-tt}}$-autoreduction for A. If $x \in f(x)$, then $x \in A$ implies the contradiction $1 = c_A(x) = 1 - \max\{c_A(y) \mid y \in f(x)\} = 1 - 1 = 0$, hence $x \notin A$. In both cases we can easily obtain a $\leq^r_{k\text{-tt}}$-autoreduction for A. The case where the number of queries is not bounded by a constant k is shown analogously. Hence item 2 and item 4 hold.

Let now A be $\leq^r_{k\text{-ctt}}$-complete for $\mathcal{C}$. From Proposition 7.4.2 we obtain that $\overline{A}$ is $\leq^r_{k\text{-dtt}}$-complete for co$\mathcal{C}$. Since $\mathcal{C}$ is closed under complement we have co$\mathcal{C} = \mathcal{C}$, hence by the previous argumentation we know that $\overline{A}$ is $\leq^r_{k\text{-tt}}$-autoreducible. From Proposition 7.4.4 we obtain that A is $\leq^r_{k\text{-tt}}$-autoreducible. The case where the number of queries is not bounded is shown analogously. Hence item 3 and item 5 hold.

Item 1 follows from item 2 for $k = 1$. □

Chapter 8

Self-Reducibility

Autoreducibility by Tracing Computation Paths We analyze the logspace autoreducibility properties of complete sets for classes of low complexity such as NL and P. Suppose we are given a logspace many-one complete set A for NL. Then there exists a nondeterministic logspace Turing machine M that accepts A. Now suppose we want to define a logspace-computable autoreduction function for A. Note that in logspace we can only memorize a constant number of configurations of M on input x. While this is not enough to store an entire computation path of M, it enables us to trace one particular computation path. Now the challenge is to trace a computation path with the correct behavior, i.e., a path that is accepting if and only if $x \in A$.

So suppose we have traced some computation path with the correct behavior down to some configuration C. If C is a stop configuration we are done, because then we know the acceptance behavior of M and hence the membership status of x to A. Otherwise there are two successor configurations C_1 and C_2 of C, and the question is whether to continue the trace with C_1 or C_2. The crucial observation here is that the question whether C_i belongs to an accepting path of M on input x or not can be answered in nondeterministic logspace and is hence reducible to A via some reduction function f. If f maps C_i to x, then we know that C_i remains on a computation path with the correct behavior, and we can continue our trace with C_i. Otherwise we obtain two queries y, z different from x such that $x \in A$ if and only if $y \in A$ or $z \in A$, and we can use y and z as the output of an autoreduction function for A.

Note that $\mathrm{P} = \mathrm{AL}$, hence for every complete set for P there exists an alternating logspace Turing machine. Again, single configurations of those machines can be stored in logspace. With minor modifications, the above technique can also be used to show autoreducibility of logspace complete sets for P.

Autoreducibility by Self-Reducibility We use the concept of self-reducibility to generalize the above scheme. Self-reducibility is a stronger form of autoreducibility, where a set is called self-reducible if it can be reduced to itself via some reduction function that maps to strictly smaller values. Our generalization will be applicable whenever a class has complete sets that are self-reducible. It is known that NL and P have self-reducible, complete sets, so our general result will in particular cover NL and P.

Contributions We summarize the contributions of this chapter as follows.

1. *Autoreducibility by Equivalent, Self-Reducible Sets.* We provide a general lemma to translate the autoreducibility property of self-reducible sets to equivalent sets. This technique is in particular applicable to classes that have self-reducible, complete sets.

2. *Autoreducibility results for* NL *and* P*.* Since NL and P have complete sets that are self-reducible, our technique provides autoreducibility of all complete sets for certain reducibility notions. Table 8.1 lists the new autoreducibility results we obtain by the self-reducibility technique in this chapter.
3. *Autoreducibility with Fixed Binary Boolean Functions.* We further consider $\leq^{\log}_{\alpha\mathrm{tt}}$-complete sets for NL and P, where α can be an arbitrary but fixed binary Boolean function. We show that for every such α, every $\leq^{\log}_{\alpha\mathrm{tt}}$-complete set for NL and P is $\leq^{\log}_{\mathrm{btt}}$-autoreducible. In a later chapter, we will obtain negative results for 3-ary Boolean functions, which make this result interesting.

reduction	NL	P
$\leq^{\log}_{\mathrm{m}}$	$\mathrm{A}^{\log}_{2\text{-dtt}}$, $\mathrm{A}^{\log}_{2\text{-ctt}}$	
$\leq^{\log}_{1\text{-tt}}$	$\mathrm{A}^{\log}_{2\text{-tt}}$	$\mathrm{A}^{\log}_{2\text{-tt}}$
$\leq^{\log}_{k\text{-dtt}}$	$\mathrm{A}^{\log}_{2k\text{-dtt}}$	
$\leq^{\log}_{\mathrm{dtt}}$	$\mathrm{A}^{\log}_{\mathrm{dtt}}$	
$\leq^{\log}_{k\text{-ctt}}$	$\mathrm{A}^{\log}_{2k\text{-ctt}}$	
$\leq^{\log}_{\mathrm{ctt}}$	$\mathrm{A}^{\log}_{\mathrm{ctt}}$	
$\leq^{\log}_{\mathrm{tt}}$	$\mathrm{A}^{\log}_{\mathrm{tt}}$	$\mathrm{A}^{\log}_{\mathrm{tt}}$

Table 8.1: Autoreducibility results shown in this chapter, where $k \geq 2$. An entry $\mathrm{A}^{\log}_{r}$ in row $\leq$ and column $\mathcal{C}$ means that every non-trivial $\leq$-complete set for $\mathcal{C}$ is $\leq^{\log}_{r}$-autoreducible.

Organization of this Chapter In Section 8.1 we give the original definition of self-reducibility for the case of $\leq^{\log}_{\mathrm{T}}$-reducibility and further define self-reducibility for the remaining reductions we consider in this chapter accordingly. In Section 8.2 we present our lemma that translates autoreducibility of self-reducible sets to equivalent sets. We apply this lemma in Section 8.3 to complete sets for NL and P and obtain new autoreducibility results. In Section 8.4 we adapt the proof technique we used for our general lemma to sets that are complete for NL or P with respect to $\leq^{\log}_{\alpha\mathrm{tt}}$, where α is an arbitrary but fixed binary Boolean function. We close this chapter with a summary and discussion in Section 8.5.

8.1 Definition of Self-Reducibility

Balcázar [Bal90] defines self-reducibility on sets as follows.

Definition 8.1.1 ([Bal90]) *A is* $\leq^{\log}_{\mathrm{T}}$-self-reducible *if there is a logspace oracle Turing machine M that accepts A with oracle A such that on input x, the queries asked by M are of the same length as x, lexicographically smaller than x, and differ from x at most in the last* $\log|x|$ *symbols.*

Note that Balcázar used an oracle access model with one oracle tape in his definition. We allow an arbitrary number of oracle tapes. Moreover, we can define self-reducibility for the reducibility notions $\leq^{\log[k]}_{\mathrm{T}}$ and $\leq^{\log}_{\mathrm{tt}}$ analogously.

Definition 8.1.2 *The notions of* $\leq_{\mathrm{T}}^{\log[k]}$-self-reducibility *and* $\leq_{\mathrm{tt}}^{\log}$-self-reducibility *are defined analogously to the case of* $\leq_{\mathrm{T}}^{\log}$*-self-reducibility, where we consider logspace oracle Turing machines with* k *oracle tapes and non-adaptive logspace oracle Turing machines with a single oracle tape, respectively.*

There is a technical difficulty in defining self-reducibility for disjunctive and conjunctive truth-table reducibilities. In these cases, the reduction cannot simply accept or reject, but has to generate queries that represent the answer. However, a self-reduction on input $x = y0^{|y|}$ is not allowed to make any query, since the last $\log|x|$ symbols of x already reached the minimal possible value. Therefore, in the definition below the self-reduction may accept or reject without asking any queries.

Definition 8.1.3 *A set* A *is called* $\leq_{\mathrm{dtt}}^{\log}$-self-reducible *if there is a logspace-computable function* f *whose values can be* 0*,* 1*, or a list of words* $\langle y_1, \dots y_n\rangle$ *where* $n \geq 1$ *such that the following holds: If* $f(x) \in \{0,1\}$*, then* $c_A(x) = f(x)$*. Otherwise, it holds that* $f(x) = \langle y_1, \dots y_n\rangle$ *such that the* y_i *are of the same length as* x*, are lexicographically smaller than* x*, differ from* x *at most in the last* $\log|x|$ *symbols, and* $c_A(x) = \max\{c_A(y_1), c_A(y_2), \cdots, c_A(y_n)\}$*. If* n *is bounded by some constant* k*, then* A *is called* $\leq_{k\text{-dtt}}^{\log}$-self-reducible*. The notions of* $\leq_{\mathrm{ctt}}^{\log}$-self-reducibility *and* $\leq_{k\text{-ctt}}^{\log}$-self-reducibility *are defined analogously, where* max *is replaced by* min*.* A *is* $\leq_{\mathrm{m}}^{\log}$-self-reducible *if it is* $\leq_{1\text{-dtt}}^{\log}$*-self-reducible.*

8.2 Autoreducibility by Self-Reducibility

Observe that self-reducibility is stronger than autoreducibility, i.e., every self-reducible set is autoreducible. We will show that in general, the autoreducibility of a self-reducible set translates to equivalent sets. We will first sketch our technique for the many-one case.

Suppose that we have two non-trivial sets A, B such that $A \equiv_{\mathrm{m}}^{\log} B$ and B is $\leq_{\mathrm{m}}^{\log}$-self-reducible. Clearly, B is $\leq_{\mathrm{m}}^{\log}$-autoreducible. We can argue that this also holds for A. Let $A \leq_{\mathrm{m}}^{\log} B$ via f, $B \leq_{\mathrm{m}}^{\log} A$ via g, and B $\leq_{\mathrm{m}}^{\log}$-self-reduces to B via h. On input x we compute $y = f(x)$, hence it holds that $c_A(x) = c_B(y)$. Let $y_0 = y$. Suppose we have already computed y_i with $c_A(x) = c_B(y_i)$. Let $z_i = g(y_i)$, hence it holds that $c_A(x) = c_B(y_i) = c_A(z_i)$. If $z_i \neq x$, we are done. Similarly, if the $\log(|y|)$ last bits of y_i are all zero, we can compute $c_B(y_i)$ and are done. Otherwise we proceed with $y_{i+1} = h(y_i)$. The algorithm is correct because $c_B(y_{i+1}) = c_B(y_i)$, and it terminates and works in logspace because y_{i+1} is lexicographically smaller than y_i and differs from y only on the last $\log(|y|)$ bits. Hence the set A is $\leq_{\mathrm{m}}^{\log}$-autoreducible as well.

Note that the algorithm we sketched above is not really useful for us, because if B is $\leq_{\mathrm{m}}^{\log}$-self-reducible, then $B \in \mathrm{L}$. However, it gives us a good intuition of how to proceed with more complicated reducibility notions. In the following lemma, we will generalize this approach to further reducibility notions, where logspace decidability does not trivially follow from self-reducibility.

Lemma 8.2.1 *Let* $l \geq 1$ *and* A, B *be non-trivial sets.*

1. *If* $A \leq_{\mathrm{m}}^{\log} B \leq_{\mathrm{tt}}^{\log} A$ *and* B *is* $\leq_{\mathrm{tt}}^{\log}$*-self-reducible, then* A *is* $\leq_{\mathrm{tt}}^{\log}$*-autoreducible.*
2. *If* $A \leq_{\mathrm{m}}^{\log} B \leq_{1\text{-tt}}^{\log} A$ *and* B *is* $\leq_{2\text{-tt}}^{\log}$*-self-reducible, then* A *is* $\leq_{2\text{-tt}}^{\log}$*-autoreducible.*
3. *If* $A \leq_{\mathrm{m}}^{\log} B \leq_{\mathrm{dtt}}^{\log} A$ *and* B *is* $\leq_{\mathrm{dtt}}^{\log}$*-self-reducible, then* A *is* $\leq_{\mathrm{dtt}}^{\log}$*-autoreducible.*
4. *If* $A \leq_{\mathrm{m}}^{\log} B \leq_{l\text{-dtt}}^{\log} A$ *and* B *is* $\leq_{2\text{-dtt}}^{\log}$*-self-reducible, then* A *is* $\leq_{2l\text{-dtt}}^{\log}$*-autoreducible.*

Proof We start with item 1. So suppose that $A \leq_{\mathrm{m}}^{\log} B$ via $f \in \mathrm{FL}$. Let h_x^q be the unary Boolean function that results from the reduction $B \leq_{\mathrm{tt}}^{\log} A$ on input q, when all queries $r \neq x$ are substituted by their answers $c_A(r)$. Hence h_x^q is the unary Boolean function with the property $c_B(q) = h_x^q(c_A(x))$. Moreover, let M be the oracle Turing machine performing the $\leq_{\mathrm{tt}}^{\log}$-self-reduction of B. Figure 8.1 describes an oracle machine that computes a $\leq_{\mathrm{T}}^{\log[1]}$-autoreduction for A on input x.

```
On input $x$:
  1. $s := f(x)$, $\beta :=$ id
  2. let $q_1, \ldots, q_k$ be the words queried by $M$ on input $s$
  3. if $h_x^{q_1}$, ..., $h_x^{q_k}$ are all constants, then return the result of $M$ on $s$,
     modified with $\beta$, where queries are answered according to $h_x^{q_1}, \ldots, h_x^{q_k}$
  4. choose the smallest $i$ such that $h_x^{q_i}$ is not constant
  5. let $s := q_i$ and $\beta := h_x^{q_i}$
  6. goto 2
```

Figure 8.1: A $\leq_{\mathrm{T}}^{\log[1]}$-autoreduction for A.

Observe that this computation can be executed in logarithmic space without querying the input x. The machine only needs logarithmic space, since the queries q_i differ from $f(x)$ only in the last $\log|x|$ symbols. Note that the computation eventually stops in line 3, since s is always replaced by a lexicographically smaller string. For the correctness note that in line 2 we always have $c_A(x) = \beta(c_B(s))$, where $c_B(s)$ equals the result of M on input s. This is true at the beginning of the computation, because f reduces A to B, and β is the identity function. Moreover, in line 4 it holds that $c_B(q_i) = h_x^{q_i}(c_A(x))$ and since $h_x^{q_i}$ is either non or id, we obtain $c_A(x) = h_x^{q_i}(c_B(q_i))$. This shows that A is $\leq_{\mathrm{T}}^{\log[1]}$-autoreducible and hence $\leq_{\mathrm{tt}}^{\log}$-autoreducible by Ladner and Lynch [LL76].

Item 2 is shown analogously. In this case, M asks at most two queries, hence $k \leq 2$. Moreover we can check whether the functions $h_x^{q_1}$ and $h_x^{q_2}$ are constants without actually answering any query (since we consider $\leq_{\text{1-tt}}^{\log}$-reductions). So by the above argumentation we have a logspace computation that eventually stops in line 3, where two non-adaptive queries different from x determine $c_A(x)$.

In the case of item 3, let $A \leq_{\mathrm{m}}^{\log} B$ via $f \in \mathrm{FL}$, let $B \leq_{\mathrm{dtt}}^{\log} A$ via $h \in \mathrm{FL}$, and let $g \in \mathrm{FL}$ be the $\leq_{\mathrm{dtt}}^{\log}$-self-reducibility function for B. We consider the algorithm described in Figure 8.2.

```
On input $x$:
  1. $y := f(x)$
  2. if $g(y) \in \{0,1\}$ then return some $z \neq x$ with $c_A(z) = g(y)$
  3. let $\langle y_1, \ldots, y_k \rangle := g(y)$
  4. if $x \notin h(y_i)$ for all $1 \leq i \leq k$ then return $\bigcup_i h(y_i)$
  5. choose the smallest $i$ such that $x \in h(y_i)$ and let $y := y_i$
  6. goto 2
```

Figure 8.2: A $\leq_{\mathrm{dtt}}^{\log}$-autoreduction for A.

First observe that this algorithm can again be executed in logarithmic space, because the functions f, g, h are logspace computable, and each y_i only differs from $f(x)$ in the last $O(\log(|x|))$ bits. Moreover, for each new iteration we replace y by a lexicographically smaller value and hence we eventually terminate. It remains to argue for the correctness of the algorithm. We first argue that before each iteration it holds that $c_A(x) = c_B(y)$. This clearly holds before the first iteration, because f reduces A to B. Now suppose this holds before some iteration in which we reach line 5. In line 5 we choose some i such that $x \in h(y_i)$. So, if $x \in A$, then $y_i \in B$, because the function h reduces B to A. Moreover, if $x \notin A$, then we already know $y \notin B$ and hence $y_i \notin B$, because g reduces B to B. Overall we obtain $c_B(y_i) = c_A(x)$, so line 5 does not change our invariant that $c_A(x) = c_B(y)$.

We have already argued that at some point the algorithm terminates. If we stop in line 2, then we return some $z \neq x$ where from the invariant it follows that $c_A(z) = g(y) = c_B(y) = c_A(x)$. If we stop in line 4, then we return $S = \bigcup_i h(y_i)$, where $x \notin S$. From the invariant we know that $c_A(x) = c_B(y)$, since g is a reduction from B to B it holds that $c_B(y) = \max\{c_B(y_1), \dots, c_B(y_k)\}$, and since h is a reduction from B to A it holds that $c_B(y_i) = \max\{c_A(z) \mid z \in h(y_i)\}$, hence $c_A(x) = \max\{c_A(z) \mid z \in S\}$. In both cases we obtain a $\leq^{\log}_{\mathrm{dtt}}$-autoreduction for A.

Item 4 is shown analogously to item 3, where we further have the observation that h maps to at most l elements and g maps to at most 2 elements, hence the set S can have at most $2l$ elements and we obtain a $\leq^{\log}_{2l\text{-dtt}}$-autoreduction. □

8.3 Applications to Low Complexity Classes

Lemma 8.2.1 translates the autoreducibility of self-reducible sets to equivalent sets. This property is particularly useful when we consider classes that have self-reducible, complete sets. In this case, Lemma 8.2.1 translates the autoreducibility of a single self-reducible, complete set to all complete sets of that class. We show that this applies to complexity classes as low as NL or P, where the characterization by nondeterministic logspace machines and alternating logspace machines shows the existence of such complete, self-reducible sets.

Lemma 8.3.1 *Let $k \geq 1$.*

1. *There is a non-trivial $\leq^{\log}_{\mathrm{m}}$-complete set for* NL *that is $\leq^{\log}_{2\text{-dtt}}$-self-reducible.*
2. *There is a non-trivial $\leq^{\log}_{\mathrm{m}}$-complete set for* P *that is $\leq^{\log}_{2\text{-tt}}$-self-reducible.*

Proof Balcázar [Bal90] considers the acyclic graph accessibility problem (AGAP, for short) that contains strings of the form $G\#s\#t$, where G is a directed graph without cycles, and s and t are vertices of the graph such that there exists a path from s to t. He notes that AGAP is $\leq^{\log}_{\mathrm{m}}$-complete for NL and $\leq^{\log}_{\mathrm{dtt}}$-self-reducible, if encoded appropriately. In the $\leq^{\log}_{\mathrm{dtt}}$-self-reduction, it suffices to check whether $s = t$ holds, or whether there exists some predecessor t' of t such that a path from s to t' exists. Note that in logspace we can transform G into a graph G' with the same properties such that each vertex has at most two incoming edges. The resulting problem is $\leq^{\log}_{\mathrm{m}}$-complete for NL and $\leq^{\log}_{2\text{-dtt}}$-self-reducible.

Analogously, Balcázar [Bal90] considers the circuit value problem (CVP, for short) that consists of strings $u\#G\#g$, where u is a binary string, G is a Boolean circuit with fan-in two and $|u|$ inputs, and g is a gate of G that evaluates to 1 under u. Ladner [Lad75] shows that CVP is $\leq^{\log}_{\mathrm{m}}$-complete for P. Balcázar [Bal90] shows that under a suitable encoding, the problem is also $\leq^{\log}_{2\text{-tt}}$-self-reducible, because g evaluates to 1 if either g is an input gate and the corresponding bit in u is set, or if the Boolean function of g applied to the values of the predecessor gates evaluates to 1. □

Theorem 8.3.2 *Let $k \geq 1$.*

1. *All non-trivial $\leq^{\log}_{\mathrm{m}}$-complete sets for* NL *are $\leq^{\log}_{2\text{-dtt}}$-autoreducible.*

2. *All non-trivial $\leq_{\text{1-tt}}^{\log}$-complete sets for* NL *and* P *are $\leq_{\text{2-tt}}^{\log}$-autoreducible.*
3. *All non-trivial $\leq_{k\text{-dtt}}^{\log}$-complete sets for* NL *are $\leq_{2k\text{-dtt}}^{\log}$-autoreducible.*
4. *All non-trivial $\leq_{\text{dtt}}^{\log}$-complete sets for* NL *are $\leq_{\text{dtt}}^{\log}$-autoreducible.*
5. *All non-trivial $\leq_{\text{tt}}^{\log}$-complete sets for* NL *and* P *are $\leq_{\text{tt}}^{\log}$-autoreducible.*

Proof Let $k \geq 1$. From Lemma 8.3.1 we obtain two non-trivial sets B and C such that B is $\leq_{\text{m}}^{\log}$-complete for NL and $\leq_{\text{2-dtt}}^{\log}$-self-reducible, and C is $\leq_{\text{m}}^{\log}$-complete for P and $\leq_{\text{2-tt}}^{\log}$-self-reducible. Let A be an arbitrary non-trivial set. We apply Lemma 8.2.1 as follows.

1. If A is $\leq_{\text{m}}^{\log}$-complete for NL, then $A \leq_{\text{m}}^{\log} B \leq_{\text{1-dtt}}^{\log} A$, hence A is $\leq_{\text{2-dtt}}^{\log}$-autoreducible.
2. If A is $\leq_{\text{1-tt}}^{\log}$-complete for P, then $A \leq_{\text{m}}^{\log} C \leq_{\text{1-tt}}^{\log} A$, hence A is $\leq_{\text{2-tt}}^{\log}$-autoreducible. If A is $\leq_{\text{1-tt}}^{\log}$-complete for NL, then $A \leq_{\text{m}}^{\log} B \leq_{\text{1-tt}}^{\log} A$, hence A is $\leq_{\text{2-tt}}^{\log}$-autoreducible.
3. If A is $\leq_{k\text{-dtt}}^{\log}$-complete for NL, then $A \leq_{\text{m}}^{\log} B \leq_{k\text{-dtt}}^{\log} A$, hence A is $\leq_{2k\text{-dtt}}^{\log}$-autoreducible.
4. If A is $\leq_{\text{dtt}}^{\log}$-complete for NL, then $A \leq_{\text{m}}^{\log} B \leq_{\text{dtt}}^{\log} A$, hence A is $\leq_{\text{dtt}}^{\log}$-autoreducible.
5. If A is $\leq_{\text{tt}}^{\log}$-complete for P, then $A \leq_{\text{m}}^{\log} C \leq_{\text{tt}}^{\log} A$, hence A is $\leq_{\text{tt}}^{\log}$-autoreducible. If A is $\leq_{\text{tt}}^{\log}$-complete for NL, then $A \leq_{\text{m}}^{\log} B \leq_{\text{tt}}^{\log} A$, hence A is $\leq_{\text{tt}}^{\log}$-autoreducible.

□

Recall that NL is closed under complementation. We hence obtain the following corollary.

Corollary 8.3.3 *Let $k \geq 1$.*

1. *All non-trivial $\leq_{\text{m}}^{\log}$-complete sets for* NL *are $\leq_{\text{2-ctt}}^{\log}$-autoreducible.*
2. *All non-trivial $\leq_{k\text{-ctt}}^{\log}$-complete sets for* NL *are $\leq_{2k\text{-ctt}}^{\log}$-autoreducible.*
3. *All non-trivial $\leq_{\text{ctt}}^{\log}$-complete sets for* NL *are $\leq_{\text{ctt}}^{\log}$-autoreducible.*

Proof Let $k \geq 1$.

1. If A is $\leq_{\text{m}}^{\log}$-complete for NL, then $\overline{A}$ is $\leq_{\text{m}}^{\log}$-complete for NL. By Theorem 8.3.2 we obtain that $\overline{A}$ is $\leq_{\text{2-dtt}}^{\log}$-autoreducible. By Proposition 7.4.4 we obtain that A is $\leq_{\text{2-ctt}}^{\log}$-autoreducible.
2. If A is $\leq_{k\text{-ctt}}^{\log}$-complete for NL, then $\overline{A}$ is $\leq_{k\text{-dtt}}^{\log}$-complete for NL by Proposition 7.4.2. By Theorem 8.3.2 we obtain that $\overline{A}$ is $\leq_{2k\text{-dtt}}^{\log}$-autoreducible. By Proposition 7.4.4 we obtain that A is $\leq_{2k\text{-ctt}}^{\log}$-autoreducible.
3. If A is $\leq_{\text{ctt}}^{\log}$-complete for NL, then $\overline{A}$ is $\leq_{\text{dtt}}^{\log}$-complete for NL by Proposition 7.4.2. By Theorem 8.3.2 we obtain that $\overline{A}$ is $\leq_{\text{dtt}}^{\log}$-autoreducible. By Proposition 7.4.4 we obtain that A is $\leq_{\text{ctt}}^{\log}$-autoreducible.

□

8.4 Reductions with Fixed Binary Boolean Functions

In the proof of Lemma 8.2.1, we always follow a reduction path that preserves information about the input. If A is complete for NL or P for the reducibility notions $\leq_{\text{m}}^{\log}$, $\leq_{\text{1-tt}}^{\log}$, or $\leq_{k\text{-dtt}}^{\log}$, then this technique shows that A is autoreducible for a fixed number of queries. Does this still holds if we turn towards $\leq_{\text{btt}}^{\log}$-completeness? In the remainder of this chapter, we will show that for fixed binary Boolean functions α we can show that $\leq_{\alpha\mathrm{tt}}^{\log}$-completeness implies $\leq_{\mathrm{btt}}^{\log}$-autoreducibility. In the next chapter, we will show that some improvements of this result are difficult to obtain.

For the remainder of this section, let $f_0^2, f_1^2, \ldots, f_{15}^2$ denote an enumeration of all binary Boolean functions such that

$$i = 2^3 \cdot f_i^2(1,1) + 2^2 \cdot f_i^2(1,0) + 2^1 \cdot f_i^2(0,1) + 2^0 \cdot f_i^2(0,0)$$

holds for all $0 \leq i \leq 15$. So, for instance, f_{15}^2 denotes the binary constant 1, while f_9^2 denotes the binary equivalence, and f_{11}^2 denotes the binary implication.

Lemma 8.4.1 *Let $\alpha = f_9^2$, hence $\alpha(x,y) = 1$ if and only if $x = y$. If A and $\leq_{\alpha\mathrm{tt}}^{\log}$-complete for* NL *or* P*, then A is $\leq_{\mathrm{btt}}^{\log}$-autoreducible.*

Proof We will argue for P (the case of NL is shown analogously). Let A be $\leq_{\alpha\mathrm{tt}}^{\log}$-complete for P, where $\alpha = f_9^2$, hence $\alpha(x,y) = 1$ if and only if $x = y$. If A is trivial, then we are done, so assume that A is non-trivial. Recall that P = AL and let M be some alternating logspace Turing machine that shows $A \in \mathrm{P}$. We consider the configuration graph of M. Let $R = \{\langle x, C\rangle \mid C$ is the root of an accepting subgraph in $M(x)\}$ and observe that $R \in \mathrm{P}$, hence $R \leq_{\alpha\mathrm{tt}}^{\log} A$ via some logspace-computable function f. We take the following assumptions.

1. For every configuration C of $M(x)$, $f(\langle x, C\rangle)$ queries two distinct words.
2. For the start configuration C_0 of $M(x)$, $f(\langle x, C_0\rangle)$ queries x.
3. For every stop configuration C_s of $M(x)$, $f(\langle x, C_s\rangle)$ does not query x.

Let us first show that these assumptions are not a restriction.

1. If $f(\langle x, C\rangle) = \langle y, y\rangle$ for some y, then $c_R(\langle x, C\rangle) = \alpha(c_A(y), c_A(y)) = 1$, so we can modify f such that $f(\langle x, C\rangle) = \langle y_1, y_2\rangle$ for two fixed elements $y_1 \neq y_2$ with $y_1, y_2 \in A$.
2. Note that $c_A(x) = c_R(\langle x, C_0\rangle)$, since C_0 is the root of $M(x)$. If we modify f such that $f(\langle x, C_0\rangle) = \langle x, y\rangle$ for some fixed $y \in A$, then we have $\alpha(c_A(x), c_A(y)) = f_9^2(c_A(x), 1) = c_A(x)$, which means that $c_R(\langle x, C_0\rangle) = \alpha(c_A(x), c_A(y))$. Hence f remains a $\leq_{\alpha\mathrm{tt}}^{\log}$-reduction from R to A.
3. Note that we know the value $c_R(\langle x, C_s\rangle)$. Choose $y, z \neq x$ such that $y \in A$ and $z \notin A$. If $c_R(\langle x, C_s\rangle) = 1$, we modify f such that $f(\langle x, C_s\rangle) = \langle y, y\rangle$, and if $c_R(\langle x, C_s\rangle) = 0$, we modify f such that $f(\langle x, C_s\rangle) = \langle y, z\rangle$. In both cases, f remains a $\leq_{\alpha\mathrm{tt}}^{\log}$-reduction from R to A.

So suppose the assumptions hold. Hence there must be a configuration C in the configuration graph of M on input x with successors C_1, C_2 such that $f(\langle x, C\rangle)$ queries x, and both $f(\langle x, C_1\rangle)$ and $f(\langle x, C_2\rangle)$ do not query x. We iterate over all configurations until we find such a configuration C. Now we can determine $c_A(x)$ by at most five queries different from x. □

Lemma 8.4.2 *Let $\alpha = f_{11}^2$, hence $\alpha(x,y) = 1$ if and only if $x = 1$ implies $y = 1$. If A is $\leq_{\alpha\mathrm{tt}}^{\log}$-complete for* NL *or* P*, then A is $\leq_{\mathrm{btt}}^{\log}$-autoreducible.*

Proof We argue for P (the case of NL is shown analogously). Let A be $\leq_{\alpha\mathrm{tt}}^{\log}$-complete for P, where $\alpha = f_{11}^2$, hence $\alpha(x,y) = 1$ if and only if $x = 1$ implies $y = 1$. If A is trivial, then we are done, so assume that A is non-trivial. Let M be some alternating logspace Turing machine that accepts A. Consider the set $R = \{\langle x, C\rangle \mid$ configuration C is the root of an accepting subgraph in $M(x)\}$ and observe that $R, \overline{R} \in \mathrm{P}$. Hence there are logspace-computable functions f, g that show $R, \overline{R} \leq_{\alpha\mathrm{tt}}^{\log} A$. Let C_0 denote the start configuration of M on input x. We take the following assumptions.

1. $f(\langle x, C_0\rangle)$ queries $\langle y, x\rangle$ for some fixed $y \in A$.
2. For every stop configuration C_s, if $f(\langle x, C_s\rangle) = \langle y, z\rangle$, then $x \notin \{y, z\}$.
3. For every stop configuration C_s, if $g(\langle x, C_s\rangle) = \langle y, z\rangle$, then $x \notin \{y, z\}$.

Let us first show that these assumptions are not a restriction.

1. Note that $c_A(x) = c_R(\langle x, C_0\rangle)$, since C_0 is the root of $M(x)$. If we modify f such that $f(\langle x, C_0\rangle) = \langle y, x\rangle$ for some fixed $y \in A$, then we have $\alpha(c_A(y), c_A(x)) = f^2_{11}(1, c_A(x)) = c_A(x)$, which means that $c_R(\langle x, C_0\rangle) = \alpha(c_A(y), c_A(x))$. Hence f remains a $\leq^{\log}_{\alpha\mathrm{tt}}$-reduction from R to A.
2. Note that we know the value $c_R(\langle x, C_s\rangle)$. Choose $y, z \neq x$ such that $y \in A$ and $z \notin A$. If $c_R(\langle x, C_s\rangle) = 1$, we modify f such that $f(\langle x, C_s\rangle) = \langle y, y\rangle$, and if $c_R(\langle x, C_s\rangle) = 0$, we modify f such that $f(\langle x, C_s\rangle) = \langle y, z\rangle$. In both cases, f remains a $\leq^{\log}_{\alpha\mathrm{tt}}$-reduction from R to A.
3. Note that we know the value $c_{\overline{R}}(\langle x, C_s\rangle)$. Choose $y, z \neq x$ such that $y \in A$ and $z \notin A$. If $c_{\overline{R}}(\langle x, C_s\rangle) = 1$, we modify g such that $g(\langle x, C_s\rangle) = \langle y, y\rangle$, and if $c_{\overline{R}}(\langle x, C_s\rangle) = 0$, we modify g such that $g(\langle x, C_s\rangle) = \langle y, z\rangle$. In both cases, g remains a $\leq^{\log}_{\alpha\mathrm{tt}}$-reduction from $\overline{R}$ to A.

We consider the algorithm described in Figure 8.3.

```
On input $x$:
 1. $C := C_0$
 2. $(C_1, C_2) :=$ successor configurations of $C$ in $M(x)$
 3. $\beta :=$ type of node $C$ in $M(x)$
 4. if $\beta$ is existential, then:
 5.    let $\langle y_1, z_1\rangle = f(\langle x, C_1\rangle)$ and $\langle y_2, z_2\rangle = f(\langle x, C_2\rangle)$
 6.    if $x \notin \{y_1, y_2, z_1, z_2\}$ then accept if $\max\{\alpha(c_A(y_1), c_A(z_1)), \alpha(c_A(y_2), c_A(z_2))\} = 1$
 7.    else if $x \in \{y_1, y_2\}$ then accept
 8.    else set $C := C_i$, where $z_i = x$, and continue with line 2
 9. if $\beta$ is universal, then:
10.    let $\langle y_1, z_1\rangle = g(\langle x, C_1\rangle)$ and $\langle y_2, z_2\rangle = g(\langle x, C_2\rangle)$
11.    if $x \notin \{y_1, y_2, z_1, z_2\}$ then accept if $\max\{\alpha(c_A(y_1), c_A(z_1)), \alpha(c_A(y_2), c_A(z_2))\} = 0$
12.    else if $x \in \{z_1, z_2\}$ then reject
13.    else set $C := C_i$, where $y_i = x$, and continue with line 2
14. reject
```

Figure 8.3: A $\leq^{\log}_{\mathrm{btt}}$-autoreduction for A.

We will show that the algorithm traverses $M(x)$ and keeps the following invariant:

$$c_A(x) = c_R(\langle x, C\rangle)$$

After line 1, the invariant clearly holds. We only change C in line 8 and in line 13. We hence distinguish the following cases.

- Assume we reach line 8. Then β is existential, and $f(\langle x, C_i\rangle) = \langle y_i, x\rangle$. Since f reduces R to A we have $x \in A \implies c_R(\langle x, C_i\rangle) = \alpha(c_A(y_i), c_A(x)) = f^2_{11}(c_A(y_i), 1) = 1$. Since β is existential it holds that $x \notin A \implies c_R(\langle x, C_i\rangle) = 0$. So after setting $C := C_i$, the invariant still holds.

- Assume we reach line 13. Then β is universal, and $g(\langle x, C_i\rangle) = \langle x, z_i\rangle$. Since g reduces $\overline{R}$ to A it holds that $x \notin A \implies c_R(\langle x, C_i\rangle) = 1 - c_{\overline{R}}(\langle x, C_i\rangle) = 1 - \alpha(c_A(x), c_A(z_i)) = 1 - f_{11}^2(0, c_A(z_i)) = 0$. Since β is universal it holds that $x \in A \implies c_R(\langle x, C_i\rangle) = 1$. So after setting $C := C_i$, the invariant still holds.

Since the invariant always holds, we can now show that the algorithm accepts correctly. We have the following cases.

- We reach line 6 and it holds that $x \notin \{y_1, y_2, z_1, z_2\}$. Here we further know that β is existential. Hence it holds that $c_A(x) = c_R(\langle x, C\rangle) = \max\{c_R(\langle x, C_1\rangle), c_R(\langle x, C_2\rangle)\} = \max\{\alpha(c_A(y_1), c_A(z_1)), \alpha(c_A(y_2), c_A(z_2))\}$, so we either accept correctly in line 6, or we reject correctly in line 14.
- We reach line 11 and it holds that $x \notin \{y_1, y_2, z_1, z_2\}$. Since β is universal it holds that $c_A(x) = c_R(\langle x, C\rangle) = \min\{c_R(\langle x, C_1\rangle), c_R(\langle x, C_2\rangle)\} = 1 - \max\{c_{\overline{R}}(\langle x, C_1\rangle), c_{\overline{R}}(\langle x, C_2\rangle)\} = 1 - \max\{\alpha(c_A(y_1), c_A(z_1)), \alpha(c_A(y_2), c_A(z_2))\}$, so we either accept correctly in line 11, or we reject correctly in line 14.
- We reach line 7 and it holds that $x = y_i$ for some $i \in \{1, 2\}$. Suppose $x \notin A$. Then we have $c_R(\langle x, C_i\rangle) = \alpha(c_A(x), c_A(z_i)) = \alpha(0, c_A(z_i)) = 1$. Since β is existential, we obtain $c_R(\langle x, C\rangle) = 1$. This contradicts the invariant, so it holds that $x \in A$, and we accept correctly.
- We reach line 12 and it holds that $x = z_i$ for some $i \in \{1, 2\}$. Suppose $x \in A$. Then we have $c_{\overline{R}}(\langle x, C_i\rangle) = \alpha(c_A(y_i), c_A(x)) = \alpha(c_A(y_i), 1) = 1$, hence $c_R(\langle x, C_i\rangle) = 0$. Since β is universal, this means that $c_R(\langle x, C\rangle) = 0$. This contradicts the invariant, so we have $x \notin A$, and we reject correctly.

Note that the algorithm will eventually terminate, because we assumed that the stop configurations do not query x. Since the algorithm queries at most four nonadaptive queries y_1, y_2, z_1, z_2, we obtain that A is $\leq_{\mathrm{btt}}^{\log}$-autoreducible. $\square$

With our previous results and the two lemmas we have shown above we can now show that for arbitrary binary Boolean functions α it holds that $\leq_{\alpha\mathrm{tt}}^{\log}$-completeness for NL or P implies $\leq_{\mathrm{btt}}^{\log}$-autoreducibility.

Corollary 8.4.3 *For every binary Boolean function α, if A is non-trivial and $\leq_{\alpha\mathrm{tt}}^{\log}$-complete for* NL *or* P*, then A is $\leq_{\mathrm{btt}}^{\log}$-autoreducible.*

Proof Note that NL and P are closed under complementation, hence we can apply Proposition 7.4.3. So if A is $\leq_{\alpha\mathrm{tt}}^{\log}$-complete for NL or P, then A is $\leq_{\overline{\alpha}\mathrm{tt}}^{\log}$-complete for NL or P, where $\overline{\alpha}\colon \{0,1\}^2 \to \{0,1\}$ such that $\overline{\alpha}(b_1, b_2) = 1 - \alpha(b_1, b_2)$ for all $b_1, b_2 \in \{0,1\}$. Note that if $\alpha = f_i^2$, then $\overline{\alpha} = f_{15-i}^2$. So it suffices to show the corollary for $\alpha \in \{f_8^2, f_9^2, \dots, f_{15}^2\}$. We distinguish the following cases.

- $\alpha \in \{f_{15}^2\}$. Then, α is constant. Since there are no $\leq_{\alpha\mathrm{tt}}^{\log}$-complete sets for NL and P with constant α, the corollary holds.
- $\alpha \in \{f_{10}^2, f_{12}^2\}$. Then, α only depends on one variable, hence A is $\leq_{1\text{-tt}}^{\log}$-complete for NL or P and thus $\leq_{\mathrm{btt}}^{\log}$-autoreducible by Theorem 8.3.2.
- $\alpha \in \{f_8^2, f_{14}^2\}$. Then, A is $\leq_{2\text{-dtt}}^{\log}$-complete or $\leq_{2\text{-ctt}}^{\log}$-complete for NL or P and thus $\leq_{\mathrm{btt}}^{\log}$-autoreducible by Proposition 7.4.5.
- $\alpha \in \{f_9^2\}$. By Lemma 8.4.1 it holds that A is $\leq_{\mathrm{btt}}^{\log}$-autoreducible.

- $\alpha \in \{f_{11}^2, f_{13}^2\}$. By symmetricity, if A is $\leq_{\alpha\mathrm{tt}}^{\log}$-complete for NL or P with $\alpha = f_{13}^2$, then A is $\leq_{\beta\mathrm{tt}}^{\log}$-complete for NL or P with $\beta = f_{11}^2$. So it suffices to consider the case where $\alpha = f_{11}^2$. By Lemma 8.4.2 we obtain that A is $\leq_{\mathrm{btt}}^{\log}$-autoreducible.

$\square$

8.5 Summary and Discussion

We started our study of logspace redundancy properties of complete sets on small classes such as NL and P. We showed that the self-reducibility property of some complete sets for these classes is very useful. Self-reducibility is a special form of autoreducibility, where the autoreduction only asks queries that are strictly smaller than the input. We proved with Lemma 8.2.1 that the autoreducibility property of a self-reducible set also holds for equivalent sets. Since NL and P have complete sets that are self-reducible, we obtained that all complete sets for these classes are autoreducible.

In the proof of Lemma 8.2.1, we followed the self-reduction graph of a self-reducible, complete set down to a point where we had values different from the input. Analogously, we can traverse the configuration graph of a nondeterministic or alternating logspace machine. We applied this technique in Section 8.4 and obtained that for arbitrary but fixed binary Boolean functions α, all $\leq_{\alpha\mathrm{tt}}^{\log}$-complete sets for NL and P are $\leq_{\mathrm{btt}}^{\log}$-autoreducible. In the next chapter we show that this is almost as far as we can hope to get: an analogous result for 3-ary Boolean functions would separate P and PSPACE (cf. Section 9.3).

Chapter 9

Local Checkability

General Approach In the last chapter we traversed the self-reduction graph of a self-reducible set until we found equivalent elements that could be used as an autoreduction. This approach worked well, because we considered self-reductions for reducibility notions such as $\leq_{\mathrm{m}}^{\log}$ or $\leq_{\mathrm{dtt}}^{\log}$ that have a very regular structure. In more complicated cases, we need additional properties.

In this chapter we consider $\leq_{\mathrm{btt}}^{\log}$-complete sets for NL, P, and the levels Δ_k^{p} of the polynomial-time hierarchy, and we show how to establish autoreducibility with respect to more complicated reducibility notions. Suppose, for instance, we want to show that a $\leq_{\mathrm{btt}}^{\log}$-complete set A for P is autoreducible. Since $\mathrm{P} = \mathrm{AL}$, there exists an alternating logspace Turing machine M that accepts A, whose configurations on input x can be described by $O(\log(|x|))$ many bits. By arguments similar to those in the last chapter there exists a configuration C with successors C_1 and C_2 that can be computed in polynomial time and that yields an autoreduction for x. However, since now the set A is only $\leq_{\mathrm{btt}}^{\log}$-complete for P, we cannot directly "navigate" from the start configuration of M on input x to C, because the reduction is too complicated.

Instead of a tree traversal we will compute C bitwise by a reduction to A. Since the query $x \in A$ is not allowed, we will use $A \cup \{x\}$ and $A - \{x\}$ as oracle sets and obtain two candidates for C. Now we locally check whether the candidate obtained from $A \cup \{x\}$ or whether the candidate obtained from $A - \{x\}$ is correct. This way we obtain access to the correct oracle, and hence we find out whether $x \in A$ holds or not. Since C consists of $O(\log(|x|))$ many bits, and since each bit is determined by a $\leq_{\mathrm{btt}}^{\log}$ reduction to A, we obtain an autoreduction that asks at most $O(\log(|x|))$ many queries.

Further Applications By a careful adaption of this technique to quantified Boolean formulas we show autoreducibility results for $\leq_{\mathrm{btt}}^{\log}$-complete sets of the levels Δ_k^{p} of the polynomial-time hierarchy. Our proof will further show that all $\leq_{\mathrm{tt}}^{\log}$-complete sets for Δ_k^{p} are $\leq_{\mathrm{tt}}^{\log}$-autoreducible. In a similar way we can locally check the transcripts of polynomial-time autoreductions. This will help us to transfer polynomial-time autoreducibility results to the logspace setting.

Contributions We summarize the contributions in this chapter as follows.

1. *New Autoreducibility Results for* NL, P *and* Δ_k^{p}. We apply local checkability and obtain new autoreducibility results for logspace complete sets for the above mentioned classes. Table 9.1 lists the new results we obtain.
2. *Translation of Polynomial-Time Autoreducibility Results to the Logspace Setting.* We locally check transcripts of polynomial-time autoreductions in logspace. For classes such as NP, there exist autoreducibility results for polynomial-time reducibility notions. By

locally checking the transcripts of polynomial-time autoreductions, we obtain new logspace autoreducibility results. Table 9.2 summarizes the results we obtain this way.

3. *Negative Results.* Buhrman et al. [BFvMT00] use diagonalization to show that there exists a $\leq^{\mathrm{p}}_{\mathrm{btt}}$-complete sets for EXP that is not $\leq^{\mathrm{p}}_{\mathrm{btt}}$-autoreducible. We adapt their proof to the logspace setting and obtain a $\leq^{\log}_{\mathrm{btt}}$-complete set for PSPACE that is not $\leq^{\log}_{\mathrm{btt}}$-autoreducible. This shows that the results we obtained for $\leq^{\log}_{\mathrm{btt}}$-complete sets for P and Δ^{p}_k are difficult to improve, because, for instance, $\leq^{\log}_{\mathrm{btt}}$-autoreducibility of all $\leq^{\log}_{\mathrm{btt}}$-complete sets for P or Δ^{p}_k would separate P or Δ^{p}_k from PSPACE.

reduction	NL	P	Δ^{p}_k
$\leq^{\log}_{\mathrm{btt}}$	$\mathrm{A}^{\log[1]}_{\text{log-T}}$	$\mathrm{A}^{\log[1]}_{\text{log-T}}$	$\mathrm{A}^{\log[1]}_{\text{log-T}}$
$\leq^{\log}_{\mathrm{tt}}$			$\mathrm{A}^{\log}_{\mathrm{tt}}$

Table 9.1: Logspace autoreducibility results shown directly in this chapter, where $k \geq 2$. An entry $\mathrm{A}^{\log}_r$ in row $\leq$ and column $\mathcal{C}$ means that every $\leq$-complete set for $\mathcal{C}$ is $\leq^{\log}_r$-autoreducible. We use the superscript log[1] to indicate that the autoreduction uses only one oracle tape.

reduction	NP	coNP	Σ^{p}_k	Π^{p}_k
$\leq^{\log}_{\mathrm{m}}, \leq^{\log}_{\text{1-tt}}, \leq^{\log}_{l\text{-dtt}}, \leq^{\log}_{\mathrm{dtt}}$	$\mathrm{A}^{\log}_{\mathrm{T}}$	$\mathrm{A}^{\log}_{\mathrm{T}}$	$\mathrm{A}^{\log}_{\mathrm{T}}$	$\mathrm{A}^{\log}_{\mathrm{T}}$

Table 9.2: Logspace autoreducibility results obtained from polynomial-time autoreducibility results, where $k, l \geq 2$. An entry $\mathrm{A}^{\log}_r$ in row $\leq$ and column $\mathcal{C}$ means that every $\leq$-complete set for $\mathcal{C}$ is $\leq^{\log}_r$-autoreducible.

Organization of this Chapter In Section 9.1 we will show direct applications of local checkability to obtain autoreducibility of $\leq^{\log}_{\mathrm{btt}}$-complete sets for the classes NL, P, and Δ^{p}_k. In Section 9.2 we will locally check transcripts of polynomial-time autoreductions to obtain logspace autoreducibility. In Section 9.3 we show a negative autoreducibility result. We conclude this chapter with a summary and discussion in Section 9.4.

9.1 Bounded Truth-Table Complete Sets

In the last chapter we have seen that for fixed binary Boolean functions α, every $\leq^{\log}_{\alpha\mathrm{tt}}$-complete set for NL and P is $\leq^{\log}_{\mathrm{btt}}$-autoreducible. We raised the question whether all arbitrary $\leq^{\log}_{\mathrm{btt}}$-complete sets for NL or P are $\leq^{\log}_{\mathrm{btt}}$-autoreducible. We will now show that for NL and P, if we allow more than a constant number of nonadaptive queries in the autoreduction, the answer is yes.

Theorem 9.1.1 *All $\leq^{\log}_{\mathrm{btt}}$-complete sets for* NL *and* P *are $\leq^{\log[1]}_{\text{log-T}}$-autoreducible.*

Proof Let A be $\leq^{\log}_{\mathrm{btt}}$-complete for P. Recall that P = AL and let M be an alternating logspace Turing machine with $L(M) = A$. Let $M(x)$ denote the configuration graph of M on input x. We consider the set $R = \{\langle x, C\rangle \mid C \text{ is the root of an accepting subgraph in } M(x)\}$. Note that $M(x)$ also contains those configurations of M on input x of length $O(\log |x|)$ that are not reachable from the start configuration of M on input x. Observe that $R \in \mathrm{AL}$, hence $R \leq_{\mathrm{btt}}^{\log} A$ via some logspace oracle Turing machine. Let h_x^C be the unary Boolean function that results from the reduction $R \leq_{\mathrm{btt}}^{\log} A$ on input $\langle x, C\rangle$, where all queries $q \neq x$ are substituted by their answers $c_A(q)$. Hence $c_R(\langle x, C\rangle) = h_x^C(c_A(x))$. We may assume that for every input x:

- $h_x^{C_0} = \mathrm{id}$, where C_0 is the start configuration of M on input x
- $h_x^{C_s}$ is constant for every stop configuration C_s of M on input x

So for every x, there exists a configuration C in $M(x)$ with successors C_1 and C_2 such that h_x^C is not constant, while $h_x^{C_1}$ and $h_x^{C_2}$ are constant. Let $C(x)$ be the smallest such configuration and let $B = \{\langle x, i\rangle \mid \text{the } i\text{-th bit of } C(x) \text{ is one}\}$. Observe that $B \in \mathrm{AL}$ and hence $B \leq_{\mathrm{btt}}^{\log} A$ via some logspace oracle Turing machine N. Further observe that $|C(x)| \in O(\log |x|)$. We consider the algorithm described in Figure 9.1.

```
On input $x$:
  1. compute $C := b_1 b_2 \dots b_{|C(x)|}$ with $b_i :=$ result of $N^{A\cup\{x\}}$ on input $\langle x, i\rangle$
  2. verify that $C$ is a configuration in $M(x)$ with two successors $C_1$ and $C_2$;
     if verification fails, then reject
  3. compute the unary Boolean functions $h_x^C$, $h_x^{C_1}$, and $h_x^{C_2}$
  4. if $h_x^C$ is constant, then reject
  5. if $h_x^{C_1}$ or $h_x^{C_2}$ is not constant, then reject
  6. if $C$ is existential and $h_x^C(h_x^{C_1} \vee h_x^{C_2})$ is true, then accept
  7. if $C$ is universal and $h_x^C(h_x^{C_1} \wedge h_x^{C_2})$ is true, then accept
  8. reject
```

Figure 9.1: A $\leq_{\text{log-T}}^{\log[1]}$-autoreduction for A.

For the correctness, we distinguish the following two cases, where x is the input.

Case 1: The algorithm does not reach line 6. Then the algorithm rejects in line 2, in line 4 or in line 5. This is only possible if $C \neq C(x)$. Hence for some i it holds that $c_B(\langle x, i\rangle)$ differs from the result of $N^{A\cup\{x\}}$ on input $\langle x, i\rangle$. Since N reduces B to A, it must hold that $A \cup \{x\} \neq A$, hence $x \notin A$, and we reject correctly.

Case 2: The algorithm reaches line 6. At this point we have computed a configuration C with successors C_1 and C_2 in $M(x)$ such that h_x^C is not constant and $h_x^{C_1}$ and $h_x^{C_2}$ are constant. Recall that $h_x^C(c_A(x)) = c_R(\langle x, C\rangle)$. If C is existential, then $c_R(\langle x, C\rangle) = (c_R(\langle x, C_1\rangle) \vee c_R(\langle x, C_2\rangle)) = (h_x^{C_1} \vee h_x^{C_2})$. If C is universal, then $c_R(\langle x, C\rangle) = (c_R(\langle x, C_1\rangle) \wedge c_R(\langle x, C_2\rangle)) = (h_x^{C_1} \wedge h_x^{C_2})$. So in either case, our algorithm accepts or rejects correctly.

Since the configurations in $M(x)$ have length $O(\log |x|)$, the algorithm can be executed in logspace. Further note that N is a $\leq_{\mathrm{btt}}^{\log}$-reduction, so if we execute N on some input, then it asks at most a constant number of queries. Since $|C(x)| \in O(\log |x|)$, in line 1 we have $O(\log |x|)$ many queries to $A \cup \{x\}$, where we do not need to query x. Moreover, in line 3, the unary Boolean functions h_x^C, $h_x^{C_1}$, and $h_x^{C_2}$ are computed by the reduction $R \leq_{\mathrm{btt}}^{\log} A$ with a constant number of queries that are different from x. Note that for all queries, one oracle tape is sufficient, and moreover, we need to ask adaptive queries, because we first have $O(\log n)$ queries to determine C and then $O(1)$ queries to compute the Boolean functions h_x^C, $h_x^{C_1}$, and $h_x^{C_2}$ in line 3. The last $O(1)$ queries depend on C and hence on the first $O(\log n)$ queries. So overall we obtain that the above algorithm is a $\leq_{\text{log-T}}^{\log[1]}$-autoreduction for A.

It remains to consider the case where A is $\leq_{\mathrm{btt}}^{\log}$-complete for NL. In this case, all configurations in $M(x)$ are existential, hence it holds that $R \in \mathrm{NL}$. Moreover we have $B \in \mathrm{NL}$. The remaining part of the proof works analogously. □

The situation for classes of higher complexity is more difficult. We will consider $\leq_{\mathrm{btt}}^{\log}$-complete sets for the levels Δ_k^{p} of the polynomial-time hierarchy. We obtain a similar autoreducibility result using more involved techniques that work with quantified satisfiability problems that are complete for the different levels of the polynomial-time hierarchy. For $z = z_1 \dots z_m \in \{0,1\}^m$ and an $(m+n)$-ary Boolean formula φ with variables $y_1, \dots, y_{m+n}$, let $\varphi(z)$ denote the n-ary Boolean formula obtained by substituting y_i for z_i where $i \in \{1, \dots, m\}$. We say that a Boolean formula is in 3-CNF if it is a conjunction of disjunctions, where each disjunction consists of 3 literals, and we say that it is in 3-DNF if it is a disjunction of conjunctions, where each conjunction consists of 3 literals.

Definition 9.1.2 *Let $k \geq 0$. If k is even, let $Q := \forall, R := \exists$, and $Q := \exists, R := \forall$ otherwise.*

1. Σ_k-3SAT $:= \{\varphi \mid \varphi$ *is a Boolean formula in 3-DNF if k is even and in 3-CNF otherwise, has km variables, and* $\exists z_k \in \{0,1\}^m \forall z_{k-1} \in \{0,1\}^m \cdots Q z_1 \in \{0,1\}^m\ \varphi(z_k, \dots, z_1) = 1\}$
2. Π_k-3SAT $:= \{\varphi \mid \varphi$ *is a Boolean formula in 3-CNF if k is even and in 3-DNF otherwise, has km variables, and* $\forall z_k \in \{0,1\}^m \exists z_{k-1} \in \{0,1\}^m \cdots R z_1 \in \{0,1\}^m\ \varphi(z_k, \dots, z_1) = 1\}$
3. Δ_k-3SAT $:= \{\varphi \in \Sigma_k$-3SAT $\mid$ *if $k \geq 1$ and φ has km variables, then the minimal $z_k \in \{0,1\}^m$ such that $\varphi(z_k) \in \Pi_{k-1}$-3SAT is even*$\}$

Note that Δ_0-3SAT $= \Sigma_0$-3SAT $= \{\varphi \mid \varphi$ is a true Boolean sentence in 3-DNF$\}$. The problems Σ_k-3SAT and Π_k-3SAT are $\leq_{\mathrm{m}}^{\log}$-complete for Σ_k^{p} and Π_k^{p}, respectively [Sto76]. We will next show that Δ_k-3SAT is $\leq_{\mathrm{m}}^{\log}$-complete for $\Delta_{k+1}^{\mathrm{p}}$.

Theorem 9.1.3 ([Wra76]) *For $k \geq 1$ and $L \in \Pi_k^{\mathrm{p}}$ there exists an $f \in \mathrm{FL}$ such that for all n, $f(2^n) = \varphi_n$ is an $(n+km)$-ary Boolean formula such that for all $x \in \{0,1\}^n$ it holds that $x \in L$ if and only if $\varphi_n(x) \in \Pi_k$-3SAT.*

Theorem 9.1.4 *For $k \geq 1$ it holds that Δ_k-3SAT is $\leq_{\mathrm{m}}^{\log}$-complete for $\Delta_{k+1}^{\mathrm{p}}$.*

Proof For every $k \geq 1$, by using binary search with access to the oracle Σ_k-3SAT, we can decide in polynomial time whether a formula φ is contained in Δ_k-3SAT, hence Δ_k-3SAT $\in \Delta_{k+1}^{\mathrm{p}}$.

To show hardness for $\Delta_{k+1}^{\mathrm{p}}$, we first consider the case where $k = 1$. Wagner [Wag87] shows that the set of all Σ_1-3SAT formulas whose maximum satisfying assignment is odd is $\leq_{\mathrm{m}}^{\mathrm{p}}$-complete for Δ_2^{p}. His proof even shows $\leq_{\mathrm{m}}^{\log}$-completeness. Note that after negating all literals in the formula, we can equivalently check whether the minimum satisfying assignment is even.

For the remainder of the proof, let $k \geq 2$. Let $A \in \Delta_{k+1}^{\mathrm{p}}$, hence $A = L(M^{\Sigma_k\text{-3SAT}})$, where M is an oracle Turing machine that runs in polynomial time. Without loss of generality we assume that M on input x asks exactly $p(|x|)$ queries, where p is some polynomial. Note that Σ_k^{p} is closed under conjunction, hence for the set

$$\Sigma_k\text{-3SAT}_\wedge := \{(H_1, \dots, H_r) \mid r \in \mathbb{N} \text{ and for all } 1 \leq i \leq r \text{ it holds that } H_i \in \Sigma_k\text{-3SAT}\}$$

it holds that Σ_k-3SAT$_\wedge \in \Sigma_k^{\mathrm{p}}$. Let Σ_k-3SAT$_\wedge \leq_{\mathrm{m}}^{\mathrm{p}} \Sigma_k$-3SAT via $g \in \mathrm{FP}$. Let q be a polynomial such that for all x, if $H_1, \dots, H_m$ are queries asked by M on input x, then $|g(H_1, \dots, H_m)| \leq q(|x|)$. We assume that $H := g(H_1, \dots, H_m)$ is of the form $H = \exists z_k \in \{0,1\}^r \forall z_{k-1} \in \{0,1\}^r \cdots Q z_1 \in \{0,1\}^r G(z_k, \dots, z_1)$ for some $r \leq q(|x|)$. For $z \in \{0,1\}^{q(|x|)}$, let $H[z]$ denote the formula we obtain by replacing z_k by the last bits in z.

Let B denote the set decided by the algorithm in Figure 9.2.

```
On input $\langle x, y, z, b\rangle$ with $y \in \{0,1\}^{p(|x|)}, z \in \{0,1\}^{q(|x|)}, b \in \{0,1\}$:
  1. simulate $M$ on $x$, where queries are answered according to the bits in $y$
  2. let $H_1, \ldots, H_m$ denote the queries that are answered positively in $y$
  3. let $H := g(H_1, \ldots, H_m)$
  4. if the simulation in line 1 rejects and $b = 1$, then reject
  5. accept if and only if $H[z] \in \Pi_{k-1}$-3SAT
```

Figure 9.2: Definition of the set B.

Claim 9.1.5 *For each x and $n = |x|$ there exists a word $yzb \in \{0,1\}^*$ with $|y| = p(n)$, $|z| = q(n)$, and $|b| = 1$, such that $\langle x, y, z, b\rangle \in B$, and for the lexicographically maximal such word it holds that $c_A(x) = b$.*

Proof Let $y^* \in \{0,1\}^{p(n)}$ correspond to the correct answers to all queries of M on input x. Hence for all $y \in \{0,1\}^{p(n)}$ with $y > y^*$ there exists some query $H_i \notin \Sigma_k$-3SAT of M on input x that is answered positively in y. So for all z and b, the algorithm on input $\langle x, y, z, b\rangle$ constructs a formula $H \notin \Sigma_k$-3SAT and hence either rejects in line 4 or in line 5, so $\langle x, y, z, b\rangle \notin B$. Now let $H_1^*, \ldots, H_m^*$ denote the queries that are answered positively in y^*. Then for all z and b, the algorithm on input $\langle x, y^*, z, b\rangle$ constructs the formula $H^* := g(H_1^*, \ldots, H_m^*) \in \Sigma_k$-3SAT. Let $z^* \in \{0,1\}^{q(n)}$ denote the largest word such that $H^*[z^*] \in \Pi_{k-1}$-3SAT. For $z > z^*$ we have $H^*[z] \notin \Pi_{k-1}$-3SAT, and the algorithm rejects, so for all b it holds that $\langle x, y^*, z, b\rangle \notin B$. We finish with the following case distinction.

- Case 1: $\langle x, y^*, z^*, 1\rangle \in B$. By the above argumentation, y^*z^*1 is the lexicographically maximal such word. Since $\langle x, y^*, z^*, 1\rangle \in B$, in the algorithm for B we did not reject in line 4. Since $b = 1$, this is only possible if the simulation in line 1 accepts. Since we used the correct answer vector, we have $c_A(x) = 1$.
- Case 2: $\langle x, y^*, z^*, 1\rangle \notin B$. In this case, in the algorithm for B we do not reach line 5, because here we would accept (recall that $H^*[z^*] \in \Pi_{k-1}$-3SAT). So we reject in line 4, which is only possible if the simulation in line 1 rejects. Since we used the correct answer vector, we have $c_A(x) = 0$. Further observe that on input $\langle x, y^*, z^*, 0\rangle$ we accept in line 5, so $\langle x, y^*, z^*, 0\rangle \in B$. So the word yzb with $y = y^*$, $z = z^*$, and $b = 0$ is lexicographically maximal with $\langle x, y, z, b\rangle \in B$.

□

Observe that $B \in \Pi^{\mathrm{p}}_{k-1}$. By Theorem 9.1.3 there exists an $f \in$ FL such that for all n, $f(2^n) = \varphi_n$ is an $(n + (k-1)m)$-ary Boolean formula such that for all $\langle x, y, z, b\rangle \in \{0,1\}^n$ it holds that $\langle x, y, z, b\rangle \in B$ if and only if $\varphi_n(\langle x, y, z, b\rangle) \in \Pi_{k-1}$-3SAT. So for every x and $n = |x|$ we obtain

$x \in A \iff$ there exists a word yzb with $\langle x, y, z, b\rangle \in B$, and for the lexicographically maximal such yzb it holds that $b = 1$

$\iff$ there exists a word yzb with $\varphi_n(\langle x, y, z, b\rangle) \in \Pi_{k-1}$-3SAT, and for the lexicographically maximal such yzb it holds that $b = 1$

$\iff$ there exists a word yzb with $\varphi_{n,x}(\langle y, z, b\rangle) \in \Pi_{k-1}$-3SAT, and for the lexicographically maximal such yzb it holds that $b = 1$

$\iff$ there exists a word yzb with $\psi_{n,x}(\langle y, z, b\rangle) \in \Pi_{k-1}$-3SAT, and for the lexicographically minimal such yzb it holds that $b = 0$

$\iff$ $\xi_{n,x} \in \Sigma_k$-3SAT, and for the lexicographically minimal yzb with $\xi_{n,x}(yzb) \in \Pi_{k-1}$-3SAT it holds that $b = 0$

$\iff \xi_{n,x} \in \Delta_k\text{-3SAT},$

where $\varphi_{n,x}$ is obtained from φ_n by substituting the variable x, $\psi_{n,x}$ is obtained from $\varphi_{n,x}$ by negating all literals that contain a variable in yzb, and $\xi_{n,x} = \exists yzb\psi_{n,x}$, where we further use dummy variables to make sure that each block of quantifiers in $\xi_{n,x}$ uses the same number of variables. Observe that $\xi_{n,x}$ can be computed from x in logspace. □

Using the completeness of Δ_k-3SAT, we now proceed to show the autoreducibility of $\leq^{\mathrm{log}}_{\mathrm{btt}}$-complete sets and $\leq^{\mathrm{log}}_{\mathrm{tt}}$-complete sets for $\Delta^{\mathrm{p}}_{k+1}$.

Theorem 9.1.6 *Let $k \geq 1$.*

1. *All $\leq^{\mathrm{log}}_{\mathrm{btt}}$-complete sets for $\Delta^{\mathrm{p}}_{k+1}$ are $\leq^{\mathrm{log}[1]}_{\text{log-T}}$-autoreducible.*
2. *All $\leq^{\mathrm{log}}_{\mathrm{tt}}$-complete sets for $\Delta^{\mathrm{p}}_{k+1}$ are $\leq^{\mathrm{log}}_{\mathrm{tt}}$-autoreducible.*

Proof Let A be $\leq^{\mathrm{log}}_{\mathrm{btt}}$-complete for $\Delta^{\mathrm{p}}_{k+1}$. Moreover, let $\mathrm{F}\Delta^{\mathrm{p}}_{k+1}$ denote the class of functions that can be computed in polynomial time with oracle access to a $\leq^{\mathrm{log}}_{\mathrm{m}}$-complete set for Σ^{p}_k.

For every $h \in \mathrm{F}\Delta^{\mathrm{p}}_{k+1}$ we define a $\leq^{\mathrm{log}}_{\mathrm{btt}}$-reduction R_h and functions $h^+, h^- \in \mathrm{F}\Delta^{\mathrm{p}}_{k+1}$ as follows: Choose the smallest $c \in \mathbb{N}$ such that $|h(x)| \leq |x|^c + c$. Let B_h be the set of pairs (x, i) such that bit i in $h(x)$'s binary representation is 1. $B_h \in \Delta^{\mathrm{p}}_{k+1}$ and hence $\leq^{\mathrm{log}}_{\mathrm{btt}}$-reduces to A via a machine R_h, where R_h is the lexicographically first such machine. So the values $R^A_h(x, i)$ for $i < |x|^c + c$ tell us the binary representation of $h(x)$. If the query x is not allowed, we can still compute the following candidates for $h(x)$:

$$h^+(x) := \sum_{i<|x|^c+c} 2^i \cdot R_h^{A\cup\{x\}}(x,i) \qquad \text{and} \qquad h^-(x) := \sum_{i<|x|^c+c} 2^i \cdot R_h^{A-\{x\}}(x,i)$$

Note that if $x \in A$, then $h^+(x) = h(x)$, and if $x \notin A$, then $h^-(x) = h(x)$.

By Theorem 9.1.4, there is an $f \in \mathrm{FL}$ and a polynomial r such that $|f(x)| < r(|x|)$ and $x \in A \iff f(x) \in \Delta_k\text{-3SAT}$ for all x. Let $\varphi_x := f(x)$. We may assume that φ_x has the right format (3-DNF if k is even, 3-CNF otherwise), has the m-bit variables $y_k, \dots, y_1$, and for all i and all $z_k, \dots, z_1 < 2^m$, the value $\varphi(z_k, \dots, z_1)$ is independent of z_i's highest bit. For every i, let $\varphi_{x,i} := \varphi_x$ if $(k - i)$ is even, and $\varphi_{x,i} := \neg\varphi_x$ otherwise. For $i = k, \dots, 1$, we define:

$$\begin{aligned} z_i(x) &:= \min(\{z < 2^m \mid \varphi_{x,i}(\overline{z}_k(x), \dots, \overline{z}_{i+1}(x), z) \in \Pi_{i-1}\text{-3SAT}\} \cup \{2^{m-1}\}) \\ s_i(x) &:= \max(\{j \leq m \mid z_i^+(x) \text{ and } z_i^-(x) \text{ differ at the } j\text{-th bit from right}\} \cup \{0\}) \\ \overline{z}_i(x) &:= \min(z_i^+(x), z_i^-(x)) \end{aligned}$$

For fixed i, $\varphi_{x,i}$ can be computed in space $O(\log(|x|))$. Moreover, observe that $z_i, s_i, \overline{z}_i \in \mathrm{F}\Delta^{\mathrm{p}}_{k+1}$. We further define the following sets.

$$\begin{aligned} F_i &:= \{x \mid \varphi_{x,i}(\overline{z}_k(x), \dots, \overline{z}_{i+1}(x)) \in \Sigma_i\text{-3SAT}\} \\ E_i &:= \{x \mid \varphi_{x,i}(\overline{z}_k(x), \dots, \overline{z}_{i+1}(x)) \in \Delta_i\text{-3SAT}\} \end{aligned}$$

Observe that $F_i, E_i \in \Delta^{\mathrm{p}}_{k+1}$ and $x \in A \iff \varphi_x \in \Delta_k\text{-3SAT} \iff x \in E_k$. So the theorem is implied by the following statement, which we show by induction:

$$F_i, E_i \leq^{\mathrm{log}[1]}_{\text{log-T}} A \text{ for } i = 0, \dots, k, \text{ where on input } x \text{ the reduction does not query } x.$$

Induction Base: Let $i = 0$. Since $E_0 = F_0$, it suffices to argue for F_0. If k is even, then $\varphi_{x,0} = \varphi_x$, and φ_x is in 3-DNF. If k is odd, then $\varphi_{x,0} = \neg\varphi_x$, and φ_x is in 3-CNF. Hence in both cases, after moving the negation to the literals, $\varphi_{x,0}$ is in 3-DNF. Define

$$s(x) := \min(\{j < r(|x|) \mid \text{conjunction } j \text{ in } \varphi_{x,0}(\overline{z}_k(x), \dots, \overline{z}_1(x)) \text{ is satisfied}\} \cup \{r(|x|)\})$$

and note that $s \in \mathrm{F}\Delta^{\mathrm{p}}_{k+1}$, hence $s^+(x)$ and $s^-(x)$ are computable in logarithmic space with $O(\log|x|)$ queries to $A - \{x\}$. So $x \in F_0 \iff \varphi_{x,0}(\overline{z}_k(x), \dots, \overline{z}_1(x)) = 1 \iff s^+(x)$ or $s^-(x)$ point to a conjunction in $\varphi_{x,0}(\overline{z}_k(x), \dots, \overline{z}_1(x))$ that is satisfied. We argue that the right-hand side of this equivalence can be tested with $O(\log|x|)$ queries to $A - \{x\}$: Both conjunctions consist of 3 literals. The value of each such literal is determined by one bit of some $\overline{z}_j(x)$. The index j and the position of these bits can be determined in logarithmic space (without oracle queries), since φ_x is computable in logarithmic space. Using $s_j^+(x)$, $s_j^-(x)$, and the reduction R_{s_j}, we can determine whether $\overline{z}_j(x) = z_j^+(x)$ or $\overline{z}_j(x) = z_j^-(x)$ (or both) holds. Hence we can also obtain the value of each bit in $\overline{z}_j(x)$ with $O(\log|x|)$ queries to $A - \{x\}$.

Induction Step: Suppose the claim holds for some $i < k$. We show the claim for $i + 1$. On input x, we determine $s_{i+1}^+(x)$ and $s_{i+1}^-(x)$ with $O(\log|x|)$ queries to $A - \{x\}$.

Case 1: $s_{i+1}^+(x) \neq s_{i+1}^-(x)$.
Without loss of generality we assume $s_{i+1}^+(x) > s_{i+1}^-(x)$. Using $R_{z_{i+1}}$ we can test with $O(\log|x|)$ queries to $A \cup \{x\}$ whether $z_{i+1}^+(x)$ and $z_{i+1}^-(x)$ differ at the $s_{i+1}^+(x)$-th bit from right. If this holds, then $s_{i+1}(x) \neq s_{i+1}^-(x)$ and $x \in A$. Otherwise, $s_{i+1}(x) \neq s_{i+1}^+(x)$ and hence $x \notin A$. Since $F_{i+1}, E_{i+1} \in \Delta^{\mathrm{p}}_{k+1}$ and we know $c_A(x)$, we can determine with $O(1)$ queries to $A - \{x\}$ whether $x \in F_{i+1}$ and whether $x \in E_{i+1}$.

Case 2: $s_{i+1}^+(x) = s_{i+1}^-(x) = s_{i+1}(x) = 0$.
In this case, $z_{i+1}(x) = z_{i+1}^+(x) = z_{i+1}^-(x)$, hence

$$x \in F_{i+1} \iff \varphi_{x,i+1}(\overline{z}_k(x), \dots, \overline{z}_{i+2}(x)) \in \Sigma_{i+1}\text{-3SAT} \iff z_{i+1}^-(x) < 2^{m-1}, \text{ and}$$
$$x \in E_{i+1} \iff \varphi_{x,i+1}(\overline{z}_k(x), \dots, \overline{z}_{i+2}(x)) \in \Delta_{i+1}\text{-3SAT} \iff z_{i+1}^-(x) \text{ is even and } < 2^{m-1}.$$

The right-hand sides correspond to bit $m - 1$ and bit 0 of $z_{i+1}^-(x)$, which can be determined via $R_{z_{i+1}}$ with a constant number of queries to $A - \{x\}$.

Case 3: $s_{i+1}^+(x) = s_{i+1}^-(x) = s_{i+1}(x) > 0$.
With $R_{z_{i+1}}$ we test with $O(\log|x|)$ queries to $A \cup \{x\}$ whether $z_{i+1}^+(x), z_{i+1}^-(x) \geq 2^{m-1}$. If so, then $z_{i+1}(x) = 2^{m-1}$, $\varphi_{x,i+1}(\overline{z}_k(x), \dots, \overline{z}_{i+2}(x)) \notin \Sigma_{i+1}$-3SAT, and $\varphi_{x,i+1}(\overline{z}_k(x), \dots, \overline{z}_{i+2}(x)) \notin \Delta_{i+1}$-3SAT, which means $x \notin F_{i+1}$ and $x \notin E_{i+1}$. Otherwise, $z_{i+1}^+(x) < 2^{m-1}$ or $z_{i+1}^-(x) < 2^{m-1}$. We can further distinguish the following cases.

- $z_{i+1}^+(x) = z_{i+1}^-(x)$.
  This is not possible, because $s_{i+1}(x) > 0$, and hence $z_{i+1}^+(x)$ and $z_{i+1}^-(x)$ are different.
- $x \in A$ and $z_{i+1}^+(x) < z_{i+1}^-(x)$.
  Then, $\overline{z}_{i+1}(x) = z_{i+1}^+(x) = z_{i+1}(x) < 2^{m-1}$, and hence $\varphi_{x,i+1}(\overline{z}_k(x), \dots, \overline{z}_{i+1}(x)) \in \Pi_i$-3SAT.
- $x \in A$ and $z_{i+1}^+(x) > z_{i+1}^-(x)$.
  Then, $z_{i+1}(x) = z_{i+1}^+(x)$, but $\overline{z}_{i+1}(x) = z_{i+1}^-(x)$, hence $\varphi_{x,i+1}(\overline{z}_k(x), \dots, \overline{z}_{i+1}(x)) \notin \Pi_i$-3SAT, because $z_{i+1}(x)$ is the smallest z with $\varphi_{x,i+1}(\overline{z}_k(x), \dots, \overline{z}_{i+2}(x), z) \in \Pi_i$-3SAT.

- $x \notin A$ and $z^+_{i+1}(x) < z^-_{i+1}(x)$.
  Then, $z_{i+1}(x) = z^-_{i+1}(x)$, but $\overline{z}_{i+1}(x) = z^+_{i+1}(x)$, hence $\varphi_{x,i+1}(\overline{z}_k(x), \dots, \overline{z}_{i+1}(x)) \notin \Pi_i$-3SAT, because $z_{i+1}(x)$ is the smallest z with $\varphi_{x,i+1}(\overline{z}_k(x), \dots, \overline{z}_{i+2}(x), z) \in \Pi_i$-3SAT.
- $x \notin A$ and $z^+_{i+1}(x) > z^-_{i+1}(x)$.
  Then, $\overline{z}_{i+1}(x) = z^-_{i+1}(x) = z_{i+1}(x) < 2^{m-1}$, and hence $\varphi_{x,i+1}(\overline{z}_k(x), \dots, \overline{z}_{i+1}(x)) \in \Pi_i$-3SAT.

From the above case distinction we obtain

$$\begin{aligned} x \in A &\iff (z^+_{i+1}(x) < z^-_{i+1}(x)) \oplus \varphi_{x,i+1}(\overline{z}_k(x), \dots, \overline{z}_{i+1}(x)) \notin \Pi_i\text{-3SAT} \\ &\iff (z^+_{i+1}(x) < z^-_{i+1}(x)) \oplus \varphi_{x,i}(\overline{z}_k(x), \dots, \overline{z}_{i+1}(x)) \in \Sigma_i\text{-3SAT} \\ &\iff (z^+_{i+1}(x) < z^-_{i+1}(x)) \oplus x \in F_i. \end{aligned}$$

Note that $z^+_{i+1}(x) < z^-_{i+1}(x)$ holds if and only if the $s_{i+1}(x)$-th bit from the right of $z^-_{i+1}(x)$ is set, which we can check using $R_{z_{i+1}}$ with $O(1)$ many queries to the oracle $A - \{x\}$. Together with the induction hypothesis, we can further test $x \in F_i$ with $O(\log |x|)$ queries to $A - \{x\}$. So we obtain $c_A(x)$ which allows us to determine with $O(1)$ queries to $A - \{x\}$ whether $x \in F_{i+1}$ and whether $x \in E_{i+1}$. This completes the induction step, hence A is $\leq^{\log[1]}_{\text{log-T}}$-autoreducible.

If we only know that A is $\leq^{\log}_{\text{tt}}$-complete for Δ^{p}_k, we can use the same proof, where we do not count the number of queries. We obtain that A is $\leq^{\log}_{\text{tt}}$-autoreducible. □

9.2 Logspace Turing Autoreducibility

We next show that in some settings, polynomial-time autoreducibility implies logspace autoreducibility. Consider for instance a logspace complete set for NP. We do not know whether NP has enough computational power to diagonalize against logspace reductions. Moreover, logspace reductions do not have enough memory to store entire computation paths of a nondeterministic polynomial-time computation. However, we know that complete sets for NP are autoreducible in the polynomial-time setting (see for instance the work of Glaßer et al. [GOP$^+$07], where the authors use the left-set technique of Ogiwara and Watanabe [OW91] to provide autoreducibility). This gives us access to deterministic polynomial-time autoreductions. In logspace we locally check the consistency of transcripts of such computations, which helps us to obtain logspace autoreducibility results. The technique that we use works for sets that are polynomial-time autoreducible and logspace Turing hard for P, and it shows new results for NP, coNP, Σ^{p}_k and Π^{p}_k.

Theorem 9.2.1 *If A is $\leq^{\log[k]}_{\text{T}}$-hard for* P *and $\leq^{\text{p}}_{\text{tt}}$-autoreducible, then A is $\leq^{\log[l]}_{\text{T}}$-autoreducible, where $l = 2k + 1$.*

Proof Since A is $\leq^{\text{p}}_{\text{tt}}$-autoreducible, there are functions $f, g \in \text{FP}$ such that for all x there exists some m such that $f(x) = \langle y_1, \dots, y_m \rangle$, $x \notin \{y_1, \dots, y_m\}$, and $c_A(x) = g(x, c_A(y_1), \dots, c_A(y_m))$.

Let M_1 be a polynomial-time Turing transducer that computes f, and let M_2 be a polynomial-time Turing transducer that computes g. We will consider the *transcripts of the Turing transducers*, which are bit string representations of the sequence of configurations of the transducer on some input, starting with the input itself, and ending on the function value computed. Given a transcript of polynomial size in n, we can verify the consistency of each bit of the transcript in space $\log(n)$ by looking at a constant number of previous bits of the transcript.

On input x, let F_x denote the transcript of M_1, and let G_x denote the transcript of M_2. We assume that there are polynomials p and q such that $|F_x| = p(|x|)$ and $|G_x| = q(|x|)$. Let c be some constant such that each bit in F_x and G_x can be verified by reading at most c previous bits in the transcript.

Let $F_x[i]$ denote bit i in F_x, and let $G_x[i]$ denote bit i in G_x. We define the sets $B := \{\langle x, i\rangle \mid F_x[i] = 1\}$ and $C := \{\langle x, i\rangle \mid G_x[i] = 1\}$. Since F_x and G_x are transcripts of polynomial-time computations, we have $B, C \in \mathrm{P}$. Since A is $\leq_{\mathrm{T}}^{\log[k]}$-hard for P, there exist logspace oracle Turing machines N_1, N_2 with k oracle tapes such that $B = L(N_1^A)$ and $C = L(N_2^A)$. We consider the algorithm described in Figure 9.3.

```
On input $x$:
 1. for $i := 1$ to $p(|x|)$:
 2.    compute $U_x[i] := N_1^{A\cup\{x\}}(\langle x, i\rangle)$
 3.    verify $U_x[i]$ by reading $U_x[j_1], U_x[j_2], \ldots, U_x[j_c]$ for some $j_1, \ldots, j_c < i$
 4.    if the verification fails, reject
 5. // here it holds that $U_x[1]U_x[2]\ldots U_x[p(|x|)] = F_x$
 6. compute $m$ such that $f(x) = \langle y_1, \ldots, y_m\rangle$ and let $y := (x, c_A(y_1), \ldots, c_A(y_m))$
 7. for $i := 1$ to $q(|y|)$:
 8.    compute $V_y[i] := N_2^{A\cup\{x\}}(\langle y, i\rangle)$
 9.    verify $V_y[i]$ by reading $V_y[j_1], V_y[j_2], \ldots, V_y[j_c]$ for some $j_1, \ldots, j_c < i$
10.    if the verification fails, reject
11. // here it holds that $V_y[1]V_y[2]\ldots V_y[q(|y|)] = G_y = G_{(x,c_A(y_1),\ldots,c_A(y_m))}$
12. if $V_y[q(|y|)] = 1$ then accept, otherwise reject
```

Figure 9.3: Autoreduction for A.

Claim 9.2.2 *The algorithm correctly decides A.*

Proof Let x be some input, and let $f(x) = \langle y_1, \ldots, y_m\rangle$ and $y = (x, c_A(y_1), \ldots, c_A(y_m))$, hence $c_A(x) = g(x, c_A(y_1), \ldots, c_A(y_m))$. We distinguish the following cases.

Case 1: $x \in A$. In this case, $A = A \cup \{x\}$, hence in line 2, in each iteration we compute $N_1^A(\langle x, i\rangle) = F_x[i]$. So in iteration i we have $U_x[j] = F_x[j]$ for all $j \leq i$, hence the verification of bit i succeeds. Hence we never reject in line 4. By a similar argumentation, we never reject in line 10. This means that we reach line 12, where $V_y = G_{(x,c_A(y_1),\ldots,c_A(y_m))}$ holds. Since $x \in A$, we have $1 = c_A(x) = g(x, c_A(y_1), \ldots, c_A(y_m)) = G_{(x,c_A(y_1),\ldots,c_A(y_m))}[q(|y|)] = V_y[q(|y|)]$, hence we accept correctly.

Case 2: $x \notin A$. The algorithm either correctly rejects in line 2 or in line 10, or it reaches line 12. In the latter case, we have verified that $U_x = F_x$ and $V_y = G_{(x,c_A(y_1),\ldots,c_A(y_m))}$. Since $x \notin A$, we have $0 = c_A(x) = g(x, c_A(y_1), \ldots, c_A(y_m)) = G_{(x,c_A(y_1),\ldots,c_A(y_m))}[q(|y|)] = V_y[q(|y|)]$, hence we reject correctly. □

Claim 9.2.3 *On input x, the algorithm can be executed in space* $\log(|x|)$ *with oracle A and* $(2k + 1)$ *oracle tapes, such that it never queries x.*

Proof Let x be some input, $n = |x|$, and let $f(x) = \langle y_1, \ldots, y_m\rangle$ and $y = (x, c_A(y_1), \ldots, c_A(y_m))$. We assume that the algorithm reaches line 12, since this argumentation includes the case where we reject earlier as well. We consider the parts of the algorithm separately.

Lines 1 to 5: The loop variable in line 1 can be stored in space $O(\log(n))$, so consider some particular iteration i. In line 2 we compute $N_1^{A\cup\{x\}}(\langle x,i\rangle)$, which is possible in space $O(\log(n))$ with oracle A and k oracle tapes by simulation of N_1 without querying x. The computed value $U_x[i]$ is verified in line 3, for which we need the values $U_x[j_1], U_x[j_2], \dots, U_x[j_c]$ for some $j_1, \dots, j_c < i$. The values $j_1, \dots, j_c$ can be computed in space $O(\log(n))$. Since $j_1, \dots, j_c < i$, the verification of $U_x[j_1], \dots, U_x[j_c]$ already succeeded in previous iterations, so we can sequentially simulate $N_1^{A\cup\{x\}}$ on $\langle x,j_1\rangle, \dots, \langle x,j_c\rangle$ to obtain $U_x[j_1], \dots, U_x[j_c]$ in logspace, again with k oracle tapes and without querying x. Since c is a constant, we can store the values $U_x[j_1], \dots, U_x[j_c]$ temporarily for the verification on the working tape, and we are not required to store the entire bit string U_x. With $U_x[j_1], \dots, U_x[j_c]$ we verify $U_x[i]$, which again works in space $O(\log(n))$.

Hence the entire loop in line 1 can be executed in space $O(\log(n))$ with oracle A and k oracle tapes, such that we never query x, and after we exit the loop it holds that $U_x = F_x$.

Note that we do not store the bit string U_x.

Line 6: Having verified that $U_x = F_x$ holds, we now have access to each single bit in F_x, say bit i, by simulating $N_1^{A\cup\{x\}}(\langle x,i\rangle)$, which takes space $O(\log(n))$ and occupies k oracle tapes. Recall that the function value $f(x)$ is encoded in the last bits of F_x. So by sequentially simulating $N_1^{A\cup\{x\}}$ on $\langle x,1\rangle, \dots, \langle x,p(|x|)\rangle$, we also have access to each single bit of $f(x) = \langle y_1, \dots, y_m\rangle$, and hence to each bit of y_j for each j. So in logspace we can compute m with k oracle tapes where we never query x.

Note that the value of m can be polynomial in n, hence again we cannot store $y = (x, c_A(y_1), \dots, c_A(y_m))$ directly on a working tape. As argued above, in space $O(\log(n))$ we can compute each bit of y_j with k oracle tapes such that we never query x. We sequentially compute each bit of y_j and copy it on the oracle tape $(k+1)$. Since f is an autoreduction, $y_j \neq x$. So after y_j is written to the oracle tape, we can query y_j and obtain $c_A(y_j)$.

Hence each bit of y can be computed in space $O(\log(n))$ with $(k+1)$ oracle tapes and without querying x.

Lines 7 to 11: The variable of the loop in line 7 can be stored in space $O(\log(n))$, so consider some particular iteration i. In line 8 we compute $N_2^{A\cup\{x\}}(\langle y,i\rangle)$. Note that we have not stored y on the working tape, but instead in space $O(\log(n))$ we have access to each bit of y, which occupies $(k+1)$ oracle tapes. Hence, by recomputing the bits of y whenever necessary, we can compute $N_2^{A\cup\{x\}}(\langle y,i\rangle)$ in space $O(\log(n))$ with oracle A and $(2k+1)$ oracle tapes by simulation of N_2 without querying x. The thus computed value $V_y[i]$ is verified in line 9, where for the verification we need the values $V_y[j_1], V_y[j_2], \dots, V_y[j_c]$ for some $j_1, \dots, j_c < i$. The values $j_1, \dots, j_c$ can be computed in space $O(\log(n))$. Since $j_1, \dots, j_c < i$, the verification of $V_y[j_1], \dots, V_y[j_c]$ already succeeded in previous iterations, so we can sequentially simulate $N_2^{A\cup\{x\}}$ on $\langle y,j_1\rangle, \dots, \langle y,j_c\rangle$ to obtain $V_y[j_1], \dots, V_y[j_c]$ in logspace, again with $(2k+1)$ oracle tapes and without querying x. In particular, since c is a constant, we can store the values $V_y[j_1], \dots, V_y[j_c]$ temporarily for the verification on the working tape, and we are not required to store the entire bit string V_y. With the values $V_y[j_1], \dots, V_y[j_c]$ we proceed to verify $V_y[i]$, which again works in space $O(\log(n))$.

Hence the entire loop in line 7 works in space $O(\log(n))$ with oracle A and $(2k+1)$ oracle tapes, such that we never query x, and after the loop it holds that $V_y = G_y$.

Again note that we do not store the bit string V_y.

Line 12: It remains to compute bit $q(|y|)$ in V_y, which again is possible in space $O(\log(n))$, with $(2k+1)$ oracle tapes and without querying x.

This means that we can execute the entire algorithm in space $O(\log(n))$ and hence in space $\log(n)$ with oracle A and $(2k+1)$ oracle tapes, such that on input x we never query x. □

From Claim 9.2.2 and Claim 9.2.3 it follows that the set A is $\leq_{\mathrm{T}}^{\mathrm{log}[2k+1]}$-autoreducible. □

Corollary 9.2.4 *If A is $\leq_{\mathrm{T}}^{\mathrm{log}}$-hard for* P *and $\leq_{\mathrm{tt}}^{\mathrm{p}}$-autoreducible, then A is $\leq_{\mathrm{T}}^{\mathrm{log}}$-autoreducible.*

Proof Let B be $\leq_{\mathrm{m}}^{\mathrm{log}}$-complete for P. Since A is $\leq_{\mathrm{T}}^{\mathrm{log}}$-hard for P, there exists some k such that $B \leq_{\mathrm{T}}^{\mathrm{log}[k]} A$. Hence for every $C \in \mathrm{P}$ it holds that $C \leq_{\mathrm{m}}^{\mathrm{log}} B \leq_{\mathrm{T}}^{\mathrm{log}[k]} A$, so A is $\leq_{\mathrm{T}}^{\mathrm{log}[k]}$-hard for P. By Theorem 9.2.1, A is $\leq_{\mathrm{T}}^{\mathrm{log}[2k+1]}$-autoreducible, hence A is $\leq_{\mathrm{T}}^{\mathrm{log}}$-autoreducible. □

We finish this section by applying Corollary 9.2.4 to sets that are complete for some complexity classes, where polynomial-time autoreducibility results are known.

Theorem 9.2.5 ([GOP+07]) *Let r be one of the reductions $\leq_{\mathrm{m}}^{\mathrm{p}}$, $\leq_{1\text{-tt}}^{\mathrm{p}}$, $\leq_{\mathrm{dtt}}^{\mathrm{p}}$, $\leq_{l\text{-dtt}}^{\mathrm{p}}$ for $l \geq 2$. Then every nontrivial set that is r-complete for one of the following classes is r-autoreducible:* PSPACE, *Σ_k^{p}, Π_k^{p}, and Δ_k^{p}.*

Note that each of the classes mentioned in Theorem 9.2.5 contains P, so here we can apply Corollary 9.2.4. For P and each further level Δ_k^{p} of the polynomial-time hierarchy we have already seen that all $\leq_{\mathrm{tt}}^{\mathrm{log}}$-complete sets are $\leq_{\mathrm{tt}}^{\mathrm{log}}$-autoreducible (see Theorem 8.3.2 and Theorem 9.1.6). Moreover, PSPACE has enough computational power to diagonalize against logspace reductions, and in the next chapter, we will even obtain mitoticity results for logspace complete sets for PSPACE. For the classes NP, coNP, and all further levels Σ_k^{p} and Π_k^{p} of the polynomial-time hierarchy, Corollary 9.2.4 and Theorem 9.2.5 provide new logspace autoreducibility results.

Corollary 9.2.6 *Let r be one of the reductions $\leq_{\mathrm{m}}^{\mathrm{log}}$, $\leq_{1\text{-tt}}^{\mathrm{log}}$, $\leq_{\mathrm{dtt}}^{\mathrm{log}}$, $\leq_{l\text{-dtt}}^{\mathrm{log}}$ for $l \geq 2$. Then every nontrivial set that is r-complete for one of the following classes is $\leq_{\mathrm{T}}^{\mathrm{log}}$-autoreducible:* NP, coNP, *Σ_k^{p}, and Π_k^{p}.*

Proof Let $\mathcal{C}$ be one of the above classes, and let A be nontrivial and r-complete for $\mathcal{C}$. Then A is $\leq_{\mathrm{T}}^{\mathrm{log}}$-hard for P, because $\mathrm{P} \subseteq \mathcal{C}$. Furthermore, A is s-complete for $\mathcal{C}$, where s is one of the reductions $\leq_{\mathrm{m}}^{\mathrm{p}}$, $\leq_{1\text{-tt}}^{\mathrm{p}}$, $\leq_{\mathrm{dtt}}^{\mathrm{p}}$, $\leq_{l\text{-dtt}}^{\mathrm{p}}$. By Theorem 9.2.5, A is s-autoreducible and hence $\leq_{\mathrm{tt}}^{\mathrm{p}}$-autoreducible. We apply Corollary 9.2.4 and obtain that A is $\leq_{\mathrm{T}}^{\mathrm{log}}$-autoreducible.

Note that A is actually $\leq_{\mathrm{T}}^{\mathrm{log}[1]}$-hard for $\mathcal{C}$, so from Theorem 9.2.1 we obtain that the Turing autoreduction uses at most three oracle tapes. □

9.3 Separation Implications

We finish this chapter by showing that there are complete sets that are not autoreducible. Buhrman et al. [BFvMT00] showed the existence of a $\leq_{3\text{-tt}}^{\mathrm{p}}$-complete set for EXP that is not $\leq_{\mathrm{btt}}^{\mathrm{p}}$-autoreducible. We adapt their proof and obtain that there exists a $\leq_{3\text{-tt}}^{\mathrm{log}}$-complete set for PSPACE that is not $\leq_{\mathrm{btt}}^{\mathrm{log}}$-autoreducible. This shows that some of the autoreducibility results we obtained so far are very difficult to improve, since such improvements imply new separation results.

Theorem 9.3.1 ([BFvMT00]) *There is a $\leq^{\mathrm{p}}_{3\text{-tt}}$-complete set for* EXP *that is not $\leq^{\mathrm{p}}_{\mathrm{btt}}$-autoreducible.*

Theorem 9.3.2 *Let $k \in \mathbb{N}$ and $p(n) = n^k$.*

1. *There is a $\leq^{\log[1]}_{2\text{-T}}$-complete set for* PSPACE *that is not $\leq^{\log}_{p(n)\text{-tt}}$-autoreducible.*
2. *There is a $\leq^{\log}_{\mathrm{btt}}$-complete set for* PSPACE *that is not $\leq^{\log}_{\mathrm{btt}}$-autoreducible.*

Proof Let $\Sigma = \{0, 1\}$, and let $M_0, M_1, \dots$ be an enumeration of all $\leq^{\log}_{p(n)\text{-tt}}$-autoreductions with the following properties:

- M_i on input x can be simulated in space $i \log(|x|)$
- M_i on input x asks queries of length at most $|x|^i$
- M_i on input x asks exactly $p(|x|)$ distinct queries

We will use the function $t\colon \mathbb{N} \to \mathbb{N}$ with $t(n) = 2^{2^{2^{2^n}}}$ and stagewise diagonalize against M_i on input $0^{t(i)}$. The choice of t will make sure that $M_i(0^{t(i)})$ only asks queries of length at most $t(i)^i < t(i+1)$. We further choose a set $K \subseteq \Sigma^*$ such that K is $\leq^{\log}_{\mathrm{m}}$-complete for PSPACE, and $K \cap \Sigma^{t(i)-1} = \emptyset$ for all $i \in \mathbb{N}$. We define $A_{-1} = \emptyset$ and iteratively construct the stages $A_0 \subseteq A_1 \subseteq A_2 \subseteq \dots$ such that $A = \bigcup_{i \in \mathbb{N}} A_i$. We will show that the set A is complete for PSPACE and not autoreducible.

Stage i of the Construction of A. Suppose we have already completed all stages $j < i$ and it holds that $\emptyset = A_{-1} \subseteq A_0 \subseteq \dots \subseteq A_{i-1}$. Let $n = t(i)$ and $x = 0^n$. We will consider M_i on input x. Let $Q = \{q_1, \dots, q_{p(n)}\}$ denote the queries of $M_i(x)$ and note that Q is independent of the oracle we use. Let

$$L = Q \cap 1\Sigma^{\geq n}$$
$$R = Q \cap 0\Sigma^{\geq n}$$

be the "left" and "right" queries in Q. Recall that for all $q \in Q$ it holds that $|q| \leq n^i$. We further subdivide L and R into $\log(i)$ many blocks by

$$L_j = L \cap \{x \mid n^{2^{j-1}} < |x| \leq n^{2^j}\}$$
$$R_j = R \cap \{x \mid n^{2^{j-1}} < |x| \leq n^{2^j}\}$$

for $1 \leq j \leq \lceil \log(i) \rceil$. This will make sure that if two words y, z are in the same block j, then we have $|z| \leq n^{2^j} = (n^{2^{j-1}})^2 < |y|^2$.

Claim 9.3.3 *At least one of the following statements is true:*

$$\forall l_1 \subseteq L_1 \exists r_1 \subseteq R_1 \forall l_2 \subseteq L_2 \exists r_2 \subseteq R_2 \dots \forall l_{\lceil \log i \rceil} \subseteq L_{\lceil \log i \rceil} \exists r_{\lceil \log i \rceil} \subseteq R_{\lceil \log i \rceil} : \quad M_i^{A_{i-1} \cup l_1 \cup l_2 \cup \dots \cup l_{\lceil \log i \rceil} \cup r_1 \cup r_2 \cup \dots \cup r_{\lceil \log i \rceil}}(x) \textit{ rejects} \tag{9.1}$$

$$\forall r_1 \subseteq R_1 \exists l_1 \subseteq L_1 \forall r_2 \subseteq R_2 \exists l_2 \subseteq L_2 \dots \forall r_{\lceil \log i \rceil} \subseteq R_{\lceil \log i \rceil} \exists l_{\lceil \log i \rceil} \subseteq L_{\lceil \log i \rceil} : \quad M_i^{A_{i-1} \cup l_1 \cup l_2 \cup \dots \cup l_{\lceil \log i \rceil} \cup r_1 \cup r_2 \cup \dots \cup r_{\lceil \log i \rceil}}(x) \textit{ accepts} \tag{9.2}$$

Proof If (9.1) does not hold, then $\exists l_1 \subseteq L_1 \forall r_1 \subseteq R_1 \dots \exists l_m \subseteq L_m \forall r_m \subseteq R_m \colon M_i^B(x)$ accepts, with $B = A_{i-1} \cup l_1 \cup l_2 \cup \dots \cup l_{\lceil \log i \rceil} \cup r_1 \cup r_2 \cup \dots \cup r_{\lceil \log i \rceil}$ and $m = \lceil \log i \rceil$. This implies $\forall r_1 \subseteq R_1 \exists l_1 \subseteq L_1 \dots \forall r_m \subseteq R_m \exists l_m \subseteq L_m \colon M_i^B(x)$ accepts. □

We finish the construction of A_i with the following case distinction.

Case 1: (9.1) holds. We define $A_i = \{x\} \cup \{1y \mid y \in K \wedge t(i) \leq |y| < t(i+1)-1\} \cup r_1 \cup \cdots \cup r_{\lceil \log i \rceil}$, where the l_j result from K and the r_j are chosen in the order $r_1, \dots, r_{\lceil \log i \rceil}$ as the lexicographically minimal $r_j \subseteq R_j$ such that $\forall l_{j+1} \subseteq L_{j+1} \exists r_{j+1} \subseteq R_{j+1} \forall l_{j+2} \subseteq L_{j+2} \cdots : M_i^{A_{i-1} \cup l_1 \cup \cdots \cup l_{\lceil \log i \rceil} \cup r_1 \cup \cdots \cup r_{\lceil \log i \rceil}}(x)$ rejects.

Case 2: (9.1) does not hold. By Claim 9.3.3 we know that (9.2) holds. We analogously define $A_i = \{0y \mid y \in K \wedge t(i) \leq |y| < t(i+1)-1\} \cup l_1 \cup \cdots \cup l_{\lceil \log i \rceil}$, where the r_j result from K and the l_j are chosen in the order $l_1, \dots, l_{\lceil \log i \rceil}$ as the lexicographically minimal $l_j \subseteq L_j$ such that $\forall r_{j+1} \subseteq R_{j+1} \exists l_{j+1} \subseteq L_{j+1} \forall l_{j+2} \subseteq L_{j+2} \cdots : M_i^{A_{i-1} \cup l_1 \cup \cdots \cup l_{\lceil \log i \rceil} \cup r_1 \cup \cdots \cup r_{\lceil \log i \rceil}}(x)$ accepts. Note that in this case, $x \notin A_i$.

We will now show that A is $\leq_{2\text{-T}}^{\log[1]}$-complete for PSPACE and not $\leq_{p(n)\text{-tt}}^{\log}$-autoreducible. We have the following claims.

Claim 9.3.4 *A is not $\leq_{p(n)\text{-tt}}^{\log}$-autoreducible.*

Proof Assume that A is $\leq_{p(n)\text{-tt}}^{\log}$-autoreducible. Then there exists a $\leq_{p(n)\text{-tt}}^{\log}$-autoreduction M_i for A. Hence it holds that $0^{t(i)} \in A \iff M_i^A(0^{t(i)})$ accepts. However, if $0^{t(i)} \in A$, then in the construction of A_i, the first case holds, and hence $M_i^A(0^{t(i)})$ rejects, which contradicts $0^{t(i)} \in A$. Analogously, if $0^{t(i)} \notin A$, then in the construction of A_i, the second case holds, and hence $M_i^A(0^{t(i)})$ accepts, which contradicts $0^{t(i)} \notin A$. In both cases we obtain a contradiction, so A is not $\leq_{p(n)\text{-tt}}^{\log}$-autoreducible. □

Claim 9.3.5 *A is $\leq_{2\text{-T}}^{\log[1]}$-hard for* PSPACE.

Proof We show that $K \leq_{2\text{-T}}^{\log[1]} A$. If $x \in \Sigma^{t(i)-1}$ for some i, we can immediately reject. So suppose $x \notin \Sigma^{t(i)-1}$ for all i. We first determine in logspace the stage i such that $t(i) \leq |x| \leq t(i+1)-2$. Next we query $0^{t(i)} \in A$. If the answer is yes, then we query $1x \in A$, and if the answer is no, then we query $0x \in A$ instead. In each case, the query belongs to the part where K is encoded, hence the reduction is correct. □

It remains to show that $A \in$ PSPACE. In order to decide A, we will often have to determine whether (9.1) or (9.2) holds. At first glance, this seems difficult. For $i \in \mathbb{N}$, let $n = t(i)$, $m = \lceil \log(i) \rceil$, and $x = 0^n$. If $M_i(x)$ asks some query q, then we only know $|q| \leq n^i$ with $i = \log^{(4)}(n)$, so we cannot write down q in polynomial space. However, in polynomial space, we can write down the length of q, the number r such that q is the r-th query asked by $M_i(x)$, and we can further determine the first bit of q. Note that $M_i(x)$ asks non-adaptive queries $q_1, \dots, q_{p(n)}$ in this order. For $u_1, v_1, \dots, u_m, v_m \in \{0,1\}^{p(n)}$, we define the set

$$\begin{aligned} B_i(u_1, v_1, \dots, u_m, v_m) := \{q_j \in \{q_1, \dots, q_{p(n)}\} \mid\ & \exists s \in \{1, \dots, m\} \text{ such that } n^{2^{s-1}} < |q_j| \leq n^{2^s} \\ & \wedge \text{ if } q_j \text{ starts with 1, then bit } j \text{ in } u_s \text{ is } 1 \\ & \wedge \text{ if } q_j \text{ starts with 0, then bit } j \text{ in } v_s \text{ is } 1\}. \end{aligned}$$

So the string u_s can be used to encode the oracle answers to all queries q_j with $n^{2^{s-1}} < |q_j| \leq n^{2^s}$ that start with 1, and v_s can be used analogously for those queries that start with 0. Hence if we want to simulate $M_i(x)$ on all possible oracle sets, it suffices to simulate $M_i(x)$ on the oracle sets $A_{i-1} \cup B_i(u_1, v_1, \dots, u_m, v_m)$, where we iterate over all bit strings $u_1, v_1, \dots, u_m, v_m$. So the statements

$$\forall u_1 \exists v_1 \forall u_2 \exists v_2 \dots \forall u_m \exists v_m \colon M_i^{A_{i-1} \cup B_i(u_1, v_1, \dots, u_m, v_m)}(0^{t(i)}) \text{ rejects} \tag{9.3}$$

$$\forall v_1 \exists u_1 \forall v_2 \exists u_2 \dots \forall v_m \exists u_m \colon M_i^{A_{i-1} \cup B_i(u_1, v_1, \dots, u_m, v_m)}(0^{t(i)}) \text{ accepts} \tag{9.4}$$

with $u_j, v_j \in \{0,1\}^{p(n)}$ for all j are equivalent to (9.1) and (9.2).

We consider the algorithm described in Figure 9.4.

```
On input $x$:
 1. let $n = |x|$
 2. if $n < t(0)$, then reject
 3. compute $i$ such that $t(i) \leq n < t(i+1)$ and let $m = \lceil \log(i) \rceil$
 4. recursively compute $A_{i-1}^* = \{q \in \Sigma^{<t(i)} \cap A \mid M_i(0^{t(i)})$ queries $q\}$
 5. if $t(i) = n$:
 6.    if $x = 0^n$, and (9.3) holds with $A_{i-1}^*$ instead of $A_{i-1}$, then accept
 7.    reject
 8. if $x = 1y$:
 9.    if (9.3) holds with $A_{i-1}^*$ instead of $A_{i-1}$, and $y \in K$, then accept
10.    if (9.3) holds with $A_{i-1}^*$ instead of $A_{i-1}$, and $y \notin K$, then reject
11.    compute $s$ such that $t(i)^{2^{s-1}} < n \leq t(i)^{2^s}$
12.    for $j := 1$ to $s$ do:
13.       compute $v_j^* \in \{0,1\}^{p(t(i))}$ such that $v_j^*$ encodes $c_K(y)$ for all $0y \in R_s$
14.       compute the smallest $u_j^*$ such that (9.4) holds with $u_l = u_l^*$ and $v_l = v_l^*$
          for all $l \leq j$ and $A_{i-1}^*$ instead of $A_{i-1}$
15.    if $x$ is the $r$-th query of $M_i(0^{t(i)})$ and the $r$-th bit in $u_s^*$ is set, accept
16. if $x = 0y$, proceed analogously to the case where $x = 1y$, with the roles of
    (9.3) and (9.4), $u_j$ and $v_j$, and $u_j^*$ and $v_j^*$ switched
17. reject
```

Figure 9.4: Decision Algorithm for A.

Claim 9.3.6 *The algorithm in Figure 9.4 correctly decides A.*

Proof By induction over i. Suppose we have the input x with $n = |x|$.

Induction Base: $n < t(0)$. In this case we reject correctly, because A does not contain words of such a small length.

Induction Step: $t(i) \leq n < t(i+1)$, and by our induction hypothesis we already know that our algorithm works correctly on inputs of length less than $t(i)$. Hence the set A_{i-1}^* is computed correctly. If $n = t(i)$ and $x = 0^n$, then $x \in A$ if and only if (9.1) holds, which is equivalent to (9.3). Observe that in (9.3) we can even replace A_{i-1} by A_{i-1}^*, because we only need to consider queries asked by $M_i(0^n)$. So if $x = 0^n$, then the algorithm either correctly accepts in line 6 or correctly rejects in line 7. Moreover, if $n = t(i)$, but $x \neq 0^n$, then $x \notin A$, and we correctly reject in line 7. So it remains to consider the case where $t(i) < n < t(i+1)$. We will only consider the case where $x = 1y$ for some y, as the other case works analogously. If $x = 1y$ and (9.3) holds, then (9.1) holds, and K is encoded into the left region of A_i, hence we accept or reject correctly in line 9 or line 10. So suppose $x = 1y$, but (9.3) and hence (9.1) do not hold. Note that in this case, (9.4) and hence (9.2) hold, so each part of the right region of A_i determines the corresponding part of the left region of A_i. Moreover, K is encoded in the right region of A_i, while the left region of A_i determines the membership of x. Here, the algorithm computes the words v_j^* that correspond to the answers to oracle queries to the right region of A_i, and for each v_j^* it computes the word u_j^* that corresponds to the answers to oracle queries to the left region of A_i. Moreover, it chooses the words u_j^* lexicographically minimal, and hence they correspond to the correct parts in A_i. This in particular means that if $x \in A_i$, then $x \in L_s$, hence x is the r-th query of $M_i(0^{t(i)})$ for some r, and hence the r-th bit in u_s^* is set and we accept. Moreover, if $x \notin A_i$, then no such bit is set (because u_s^* was chosen minimal), and we reject correctly. □

Claim 9.3.7 *The algorithm in Figure 9.4 works in polynomial space.*

Proof Let p' denote a polynomial such that $K \in \mathrm{DSPACE}(p')$. We show that there exists a polynomial g such that the algorithm works in space $g(n)$ by induction over i. So suppose we work on input x with $n = |x|$.

Induction Base: If $n < t(0)$, then we reject. This can be done in constant space, because the test $n < t(0)$ is a finite case distinction.

Induction Step: Suppose we have $t(i) \leq n < t(i+1)$ for some i, and on inputs of length less than $t(i)$, the algorithm works in space g. We reserve the following space for the execution on x:

- $g(n-1)$ to recursively decide A_{i-1}
- $2m \cdot p(t(i)) \leq n \cdot p(n)$ to write down the strings $u_1, v_1, \ldots, u_m, v_m$
- $m \cdot p(t(i)) \leq n \cdot p(n)$ to write down the set A_{i-1}^*
- $2 \cdot i \cdot \log(|x|) \leq n$ to simulate $M_i(0^{t(i)})$ twice in parallel
- $p'(n^2)$ to decide K on inputs of length at most n^2
- $n \cdot p(n)$ to decide $q \in B_i(u_1, v_1, \ldots, u_m, v_m)$

Within the reserved space, we can decide whether (9.3) or (9.4) holds by iterating over all $u_1, v_1, \ldots, u_m, v_m$, simulating $M_i(0^{t(i)})$, and answering queries either according to A_{i-1}^* or to $B_i(u_1, v_1, \ldots, u_m, v_m)$. Note that this can be done by simulating $M_i(0^{t(i)})$ twice in parallel, once to find out whether $M_i(0^{t(i)})$ accepts or rejects, and once to decide $B_i(u_1, v_1, \ldots, u_m, v_m)$ on queries. Further note that we need to decide K on inputs of length at most n^2, because if $n = t(i)^{2^{s-1}} + 1$, then when we determine the bits of v_s^*, we have to compute $c_K(y)$ with $|y| \leq t(i)^{2^s} = t(i)^{2^{s-1} \cdot 2} = (t(i)^{2^{s-1}})^2 \leq n^2$. So there exists a polynomial h such that in the induction step we need space at most $g(n-1) + h(n)$. If we further choose h to be monotone and g such that $g(n) = n \cdot h(n)$, then we obtain that in the induction step we also need space at most $g(n-1) + h(n) = (n-1) \cdot h(n-1) + h(n) \leq (n-1) \cdot h(n) + h(n) = n \cdot h(n) = g(n)$. This proves the claim. □

From the last two claims we obtain $A \in \mathrm{PSPACE}$. Moreover, note that A is $\leq_{\mathrm{btt}}^{\log}$-complete and not $\leq_{\mathrm{btt}}^{\log}$-autoreducible. This finishes the proof. □

Theorem 8.3.2 states that all $\leq_{1\text{-tt}}^{\log}$-complete sets for NL and P are $\leq_{\mathrm{btt}}^{\log}$-autoreducible. Moreover, Corollary 8.4.3 states that all $\leq_{\alpha\mathrm{tt}}^{\log}$-complete sets for NL and P are $\leq_{\mathrm{btt}}^{\log}$-autoreducible, where α is an arbitrary but fixed binary Boolean function. On the other hand, Theorem 9.3.2

states that there exists a set A that is $\leq^{\log}_{3\text{-tt}}$-complete for PSPACE and not $\leq^{\log}_{\text{btt}}$-autoreducible. From the proof it even follows that A is $\leq^{\log}_{\alpha\text{tt}}$-complete for a fixed 3-ary Boolean function α. Consequently, generalizing Corollary 8.4.3 to 3-ary Boolean functions is difficult, because such an improvement shows $A \notin \text{P}$ and hence separates P and PSPACE.

- If all $\leq^{\log}_{3\text{-tt}}$-complete sets for P are $\leq^{\log}_{\text{btt}}$-autoreducible, then $\text{P} \neq \text{PSPACE}$.

On the other hand, Theorem 9.1.1 and Theorem 9.1.6 show that all $\leq^{\log}_{\text{btt}}$-complete sets for P and the levels Δ^{p}_k of the polynomial-time hierarchy are $\leq^{\log[1]}_{\log\text{-T}}$-autoreducible. Showing $\leq^{\log}_{\text{btt}}$-autoreducibility of these sets is again difficult, as it implies that $A \notin \Delta^{\text{p}}_k$ or $A \notin \text{P}$ and hence again separates P or even Δ^{p}_k and PSPACE.

- If all $\leq^{\log}_{3\text{-tt}}$-complete sets for Δ^{p}_k are $\leq^{\log}_{\text{btt}}$-autoreducible, then $\Delta^{\text{p}}_k \neq \text{PSPACE}$.

We obtain a particularly interesting situation, because every nontrivial set in L is trivially $\leq^{\log}_{\text{m}}$-autoreducible and even $\leq^{\log}_{\text{m}}$-mitotic. Since we now have positive results for L and negative results for PSPACE, settling the question whether all complete sets of intermediate classes are autoreducible or mitotic either way would yield new separation results.

Corollary 9.3.8 *Let $k \geq 1$ and $\mathcal{C}$ be any of the complexity classes* P, NP, Σ^{p}_k, Π^{p}_k, Δ^{p}_k.

1. *If all $\leq^{\log}_{\text{btt}}$-complete sets for $\mathcal{C}$ are $\leq^{\log}_{\text{btt}}$-autoreducible, then $\mathcal{C} \neq \text{PSPACE}$, otherwise $\text{L} \neq \mathcal{C}$.*
2. *If all $\leq^{\log}_{\text{btt}}$-complete sets for $\mathcal{C}$ are $\leq^{\log}_{\text{btt}}$-mitotic, then $\mathcal{C} \neq \text{PSPACE}$, otherwise $\text{L} \neq \mathcal{C}$.*

Proof By Theorem 9.3.2 there exists a set A that is $\leq^{\log}_{\text{btt}}$-complete for PSPACE and not $\leq^{\log}_{\text{btt}}$-autoreducible. If all $\leq^{\log}_{\text{btt}}$-complete sets for $\mathcal{C}$ are $\leq^{\log}_{\text{btt}}$-mitotic, then all $\leq^{\log}_{\text{btt}}$-complete sets for $\mathcal{C}$ are $\leq^{\log}_{\text{btt}}$-autoreducible, and if all $\leq^{\log}_{\text{btt}}$-complete sets for $\mathcal{C}$ are $\leq^{\log}_{\text{btt}}$-autoreducible, then A is not $\leq^{\log}_{\text{btt}}$-complete for $\mathcal{C}$ and hence $\mathcal{C} \neq \text{PSPACE}$. If there exists a set B that is $\leq^{\log}_{\text{btt}}$-complete for $\mathcal{C}$ and not $\leq^{\log}_{\text{btt}}$-autoreducible, then B is not $\leq^{\log}_{\text{btt}}$-mitotic, and if there exists a set B that is $\leq^{\log}_{\text{btt}}$-complete for $\mathcal{C}$ and not $\leq^{\log}_{\text{btt}}$-mitotic, then $B \notin \text{L}$ and hence $\text{L} \neq \mathcal{C}$. □

9.4 Summary and Discussion

We showed how to use local checkability to prove autoreducibility of complete sets. We generally used objects such as transcripts of computations, configurations of logspace machines, or quantified Boolean formulas, whose consistency can be checked in logspace by local computations. We used the key observation that single bits of these objects can be reduced to the complete set A. Since autoreductions are not allowed to query the input x, we computed two candidate objects from the oracle sets $A \cup \{x\}$ and $A - \{x\}$. By locally checking the consistency of the candidate objects, we obtained the correct object and hence the correct oracle set without querying x.

We obtained that all $\leq^{\log}_{\text{btt}}$-complete sets for NL and P are $\leq^{\log[1]}_{\log\text{-T}}$-autoreducible, and with some more effort, we were able to show the same result for each level Δ^{p}_k of the polynomial-time hierarchy. Moreover, we proved that in some cases, polynomial-time autoreducibility implies logspace autoreducibility, where we locally checked the transcripts of the polynomial-time autoreductions in logspace. This was particularly helpful for logspace complete sets for NP, coNP, and the remaining levels Σ^{p}_k and Π^{p}_k of the polynomial-time hierarchy, where we were able to show the first autoreducibility results in the logspace setting.

We further adapted a result of Buhrman et al. [BFvMT00] to the logspace setting and obtained that there exists a $\leq_{\mathrm{btt}}^{\log}$-complete set for PSPACE that is not $\leq_{\mathrm{btt}}^{\log}$-autoreducible. This shows that improvements of the above mentioned autoreducibility results for P or Δ_k^{p} to a constant number of queries are very difficult to obtain. Such improvements would imply that all complete sets for P or Δ_k^{p} share a property that some sets in PSPACE do not have, and thus separate PSPACE from P or even from Δ_k^{p}.

Chapter 10

Redundancy by Diagonalization

Diagonalization and Enforced Properties Complexity classes such as PSPACE, EXP, and NEXP have enough computational power to simulate arbitrary logspace reductions. We define complete sets for these classes that simulate reductions and enforce certain properties. Buhrman [Buh93] and Homer et al. [HKR93] use this technique to show that all $\leq^{\mathrm{p}}_{1\text{-tt}}$-complete sets for EXP and NEXP are $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete. In a similar way we can make sure that reductions do not need to query their own input, which can be used to show autoreducibility. We sketch this technique more precisely as follows.

Consider a $\leq^{\log}_{\mathrm{m}}$-complete set A for PSPACE, and suppose we want to show that A is $\leq^{\log}_{\mathrm{m}}$-autoreducible. Let $f_0, f_1, \ldots$ be an enumeration of all $\leq^{\log}_{\mathrm{m}}$-reductions such that f_i on input x can be simulated in space $i \cdot \log(1 + |x|)$. We define a set B by the algorithm that is described in Figure 10.1.

```
On input $\langle x, 0^i \rangle$:
 1. let $y := f_i(\langle x, 0^i \rangle)$
 2. if $y = x$, then reject
 3. accept if $x \in A$, and reject otherwise
```

Figure 10.1: Diagonalization enforces autoreducibility properties.

Note that $B \in$ PSPACE, so $B \leq^{\log}_{\mathrm{m}} A$ via some reduction function f_j. On input x, we consider the value $y = f_j(\langle x, 0^j \rangle)$. If $y = x$, then we obtain $0 = c_B(\langle x, 0^j \rangle) = c_A(f_j(\langle x, 0^j \rangle)) = c_A(x)$ and hence we know $x \notin A$. If $y \neq x$, then we obtain $c_A(x) = c_B(\langle x, 0^j \rangle) = c_A(f_j(\langle x, 0^j \rangle)) = c_A(y)$, so y can be used as an autoreduction value for x. So for the function $g(x) := f_j(\langle x, 0^j \rangle)$ we know $c_A(x) = c_A(g(x))$, and we enforced that $g(x) = x$ implies $x \notin A$. This property enables us to turn g into an autoreduction for A.

More General Applications In many cases, we can apply diagonalization to obtain even stronger results. For complete sets of classes that satisfy some closure properties, we use diagonalization to obtain logspace autoreductions that are length-increasing, and we show that standard techniques even imply mitoticity. We further show that a similar technique applies to reducibility notions that have a very regular structure, such as $\leq^{\log}_{\mathrm{dtt}}$ and $\leq^{\log}_{\mathrm{ctt}}$.

Contributions We summarize our contributions in this chapter as follows.

1. *General Results.* We show results that can generally be applied to complexity classes that have enough computational power to simulate logspace reductions and that further

have some additional closure properties. We show that if $\mathcal{C}$ is a complexity class that is closed under union, complement, and $\leq_{\mathrm{m}}^{\log^2\text{-lin}}$-reducibility, then its $\leq_{\mathrm{m}}^{\log}$-complete sets are $\leq_{\mathrm{m}}^{\log}$-mitotic. This result is obtained using diagonalization in a way that enforces length-squaring autoreductions. We further show that if $\mathcal{C}$ is closed under $\leq_{\text{1-tt}}^{\log^2\text{-lin}}$-reducibility, then all $\leq_{\text{2-tt}}^{\log}$-complete sets for $\mathcal{C}$ are $\leq_{\text{2-tt}}^{\log}$-autoreducible.

2. *Results for* PSPACE*,* EXP*,* NEXP*.* We consider different reducibility notions. For PSPACE and EXP, we use diagonalization to enforce that complete sets are equivalent with respect to length-increasing reductions. This enables us to show mitoticity of complete sets. Since we do not know whether NEXP is closed under complementation or not, the situation here is more complicated. We will use diagonalization to enforce that for various reducibility notions, complete sets for NEXP are autoreducible. Table 10.1 summarizes the new results we obtain for PSPACE, EXP and NEXP.

reduction	PSPACE	EXP	NEXP
$\leq_{\mathrm{m}}^{\log}, \leq_{\text{1-tt}}^{\log}, \leq_{k\text{-ctt}}^{\log}, \leq_{k\text{-dtt}}^{\log}, \leq_{\mathrm{ctt}}^{\log}, \leq_{\mathrm{dtt}}^{\log}, \leq_{\text{2-tt}}^{\log}$	$\mathrm{M}^{\log}$	$\mathrm{M}^{\log}$	$\mathrm{A}^{\log}$
$\leq_{k\text{-ctt}}^{\mathrm{p}}, \leq_{k\text{-dtt}}^{\mathrm{p}}, \leq_{\mathrm{ctt}}^{\mathrm{p}}, \leq_{\mathrm{dtt}}^{\mathrm{p}}, \leq_{\text{2-tt}}^{\mathrm{p}}$		M^{p}	A^{p}

Table 10.1: Autoreducibility and mitoticity results shown in this chapter. Let $s \in \{\mathrm{p}, \log\}$. An entry A^s in row $\leq_r^s$ and column $\mathcal{C}$ means that every $\leq_r^s$-complete set for $\mathcal{C}$ is $\leq_r^s$-autoreducible, while the entry M^s means that every $\leq_r^s$-complete set for $\mathcal{C}$ is $\leq_r^s$-mitotic.

Organization of this Chapter In Section 10.1 we show the mitoticity and autoreducibility results that generally apply to complete sets for classes that are powerful enough. In Section 10.2 we show mitoticity results for the classes PSPACE and EXP and various logspace and polynomial-time reducibility notions. In Section 10.3 we consider complete sets for NEXP. We conclude the chapter with a summary and discussion in Section 10.4.

10.1 General Results

We first consider general classes that have powerful closure properties. We define the following reducibility notions.

Definition 10.1.1 *For sets A and B we define $A \leq_{\mathrm{m}}^{\log^2\text{-lin}} B$ if and only if there exists a function $f \in \mathrm{FSPACE}(\log(n)^2)$ and some $c \in \mathbb{N}$ such that for all x it holds that $|f(x)| \leq c \cdot |x|$ and $c_A(x) = c_B(f(x))$.*

Definition 10.1.2 *For sets A and B we define $A \leq_{\text{1-tt}}^{\log^2\text{-lin}} B$ if and only if there exists an oracle Turing machine M such that $A = L(M^B)$ that uses at most $\log(n)^2$ space and asks at most one oracle question of length at most $c \cdot n$, where c is some constant.*

Suppose a class $\mathcal{C}$ is closed under union, complement, and $\leq_{\mathrm{m}}^{\log^2\text{-lin}}$-reducibility, and we have a set $A \in \mathcal{C}$. Being closed under $\leq_{\mathrm{m}}^{\log^2\text{-lin}}$-reducibility ensures that we can define another set B that simulates $\leq_{\mathrm{m}}^{\log}$-reductions to A and still remains in $\mathcal{C}$. In the definition of B we look at these simulations and construct B in such a way that only length-increasing reductions to A are possible. If A is complete for $\mathcal{C}$, then B can be reduced to A via a length-increasing reduction. We further make sure that B contains information about A and hence obtain a length-increasing reduction from A to B and back to A.

Theorem 10.1.3 *If the class $\mathcal{C}$ is closed under union, complement, and $\leq_{\mathrm{m}}^{\log^2\text{-lin}}$-reducibility, then all $\leq_{\mathrm{m}}^{\log}$-complete sets for $\mathcal{C}$ are $\leq_{\mathrm{m}}^{\log}$-autoreducible via some $f \in \mathrm{FL}$ such that $|f(x)| \geq |x|^2$ for all x.*

Proof Let A be $\leq_{\mathrm{m}}^{\log}$-complete for $\mathcal{C}$. Let $f_1, f_2, \ldots$ be an enumeration of all logspace bounded Turing transducers such that the computation of f_i on x can be simulated in space $\log i \cdot (1+\log|x|)$ using a binary alphabet. We consider the following sets.

$$B_1 = \{y \mid y = \langle 0^i, x, 0^{|x|^2}\rangle,\ |f_i(y)| \leq |y|, \text{ and } f_i(y) \notin A\}$$
$$B_2 = \{y \mid y = \langle 0^i, x, 0^{|x|^2}\rangle,\ |f_i(y)| > |y|, \text{ and } x \in A\}$$

Note that $B_1 \leq_{\mathrm{m}}^{\log^2\text{-lin}} \overline{A}$ by first checking if $y = \langle 0^i, x, 0^{|x|^2}\rangle$ and then computing $f_i(y)$ in space $\log(i) \cdot (1 + \log(|y|)) \leq \log(|y|) \cdot (1 + \log(|y|)) \in O(\log(n)^2)$ if the length does not exceed $|y|$. Since $\mathcal{C}$ is closed under complementation we have $\overline{A} \in \mathcal{C}$, and since $\mathcal{C}$ is closed under $\leq_{\mathrm{m}}^{\log^2\text{-lin}}$-reducibility we have $B_1 \in \mathcal{C}$. Analogously we show $B_2 \leq_{\mathrm{m}}^{\log^2\text{-lin}} A$ by first checking the form of y, then computing $f_i(y)$ in space $O(\log(n)^2)$ until we know that $|f_i(y)| > |y|$, and finally returning x. So we also obtain that $B_2 \in \mathcal{C}$.

Let $B = B_1 \cup B_2$. Since $\mathcal{C}$ is closed under union, we obtain $B \in \mathcal{C}$. Hence we have $B \leq_{\mathrm{m}}^{\log} A$ via some $f_j \in \mathrm{FL}$. If there exists some $y = \langle 0^j, x, 0^{|x|^2}\rangle$ such that $|f_j(y)| \leq |y|$, then we have $y \in B \iff y \in B_1 \iff f_j(y) \notin A$, which contradicts the fact that f_j reduces B to A. Hence for all y of the form $y = \langle 0^j, x, 0^{|x|^2}\rangle$ we have $|f_j(y)| > |y|$. Therefore we obtain

$$c_A(x) = c_{B_2}(\langle 0^j, x, 0^{|x|^2}\rangle) = c_B(\langle 0^j, x, 0^{|x|^2}\rangle) = c_A(f_j(\langle 0^j, x, 0^{|x|^2}\rangle)),$$

where $|f_j(\langle 0^j, x, 0^{|x|^2}\rangle)| > |\langle 0^j, x, 0^{|x|^2}\rangle| > |x|^2$. □

Next suppose we have an autoreducible set whose autoreduction function at least squares the length of its values. We subdivide the elements of the autoreducible set by their length into stages that are large enough such that for each element we can find an equivalent element that is contained in the next stage. We partition the autoreducible set into even and odd numbered stages and obtain two equivalent sets. This shows that the autoreducible set is even mitotic.

Lemma 10.1.4 *If A is $\leq_{\mathrm{m}}^{\log}$-autoreducible via some $f \in \mathrm{FL}$ such that for all x it holds that $|f(x)| \geq |x|^2$, then A is $\leq_{\mathrm{m}}^{\log}$-mitotic.*

Proof Since $f \in \mathrm{FL}$ there exists some $c \geq 3$ such that $|x|^2 \leq |f(x)| < |x|^c$ for all $x \in \Sigma^{\geq 2}$. Let

$$S = \{x \mid \min\{i \in \mathbb{N} \mid |x| \leq 2^{c^i}\} \text{ is even }\}$$

and note that $S \in \mathrm{L}$. Observe that for every x with $2^{c^{i-1}} < |x| \leq 2^{c^i}$ and $j = \lceil \log(c) \rceil$ it holds that

$$2^{c^{i+1}} \geq |x|^c > |f(x)| \qquad \text{and} \qquad |f^{(j)}(x)| \geq |x|^{2^j} \geq |x|^{2^{\log(c)}} = |x|^c > \left(2^{c^{i-1}}\right)^c = 2^{c^i}$$

and so for every x there exists some $j \in \{1, \ldots, \lceil \log(c) \rceil\}$ such that $c_S(x) \neq c_S(f^{(j)}(x))$. Let

$$r(x) = \begin{cases} f^{(i)}(x) & \text{if } |x| \geq 2 \text{ and } i = \min\{j \in \{1, \ldots, \lceil \log(c) \rceil\} \mid c_S(x) \neq c_S(f^{(j)}(x))\}\ , \\ a_1 & \text{if } |x| < 2 \text{ and } x \in A, \text{ and} \\ a_0 & \text{if } |x| < 2 \text{ and } x \notin A, \end{cases}$$

where $a_1 \in A$ and $a_0 \notin A$ are fixed elements with $a_1 \notin S$ and $a_0 \notin S$. Note that $r \in \mathrm{FL}$, since c is constant, $S \in \mathrm{L}$ and $f \in \mathrm{FL}$. Since f is a $\leq_{\mathrm{m}}^{\log}$-autoreduction for A we have $c_A(x) = c_A(r(x))$ for all x. If $|x| \geq 2$, then $c_S(x) \neq c_S(r(x))$, and if $|x| < 2$, then $c_S(x) = 1$ and $c_S(r(x)) = c_S(a_1) = c_S(a_0) = 0$. So for all x we have $c_A(x) = c_A(r(x))$ and $c_S(x) \neq c_S(r(x))$, which shows that $A \cap S \leq_{\mathrm{m}}^{\log} A \cap \overline{S}$ via r and $A \cap \overline{S} \leq_{\mathrm{m}}^{\log} A \cap S$ via r. We further have $A \leq_{\mathrm{m}}^{\log} A \cap S$, because on input x we can check in logspace whether $r(x) \in S$ holds and either use x or $r(x)$ for the reduction. The reduction $A \cap S \leq_{\mathrm{m}}^{\log} A$ works similarly by first checking whether $x \in S$. If $x \notin S$, we return a_0, otherwise we return x. This shows that A is $\leq_{\mathrm{m}}^{\log}$-mitotic. □

Corollary 10.1.5 *If the class $\mathcal{C}$ is closed under union, complement, and $\leq_{\mathrm{m}}^{\log^2\text{-lin}}$-reducibility, then all $\leq_{\mathrm{m}}^{\log}$-complete sets for $\mathcal{C}$ are $\leq_{\mathrm{m}}^{\log}$-mitotic.*

Proof Let A be $\leq_{\mathrm{m}}^{\log}$-complete for $\mathcal{C}$. By Theorem 10.1.3, A is $\leq_{\mathrm{m}}^{\log}$-autoreducible via some autoreduction function $f \in \mathrm{FL}$ such that $|f(x)| \geq |x|^2$ for all x. By Lemma 10.1.4, A is $\leq_{\mathrm{m}}^{\log}$-mitotic. □

Corollary 10.1.6 *All $\leq_{\mathrm{m}}^{\log}$-complete sets for the following classes are $\leq_{\mathrm{m}}^{\log}$-mitotic:* $\mathrm{QP} = \mathrm{DTIME}(2^{\mathrm{polylog}(n)})$, PSPACE, EXP, REC, $\mathrm{DSPACE}(s)$ *and* $\mathrm{NSPACE}(s)$ *for all space-constructible $s \geq \log^2$.*

Proof The classes satisfy the requirements in Corollary 10.1.5. □

Corollary 10.1.7 *All $\leq_{1\text{-tt}}^{\log}$-complete sets for* PSPACE *and* EXP *are $\leq_{\mathrm{m}}^{\log}$-mitotic.*

Proof Buhrman [Buh93] and Homer et al. [HKR93] showed that $\leq_{1\text{-tt}}^{\mathrm{p}}$-complete sets for EXP are $\leq_{\mathrm{m}}^{\mathrm{p}}$-complete. Their proof also shows that all $\leq_{1\text{-tt}}^{\log}$-complete sets for EXP and PSPACE are $\leq_{\mathrm{m}}^{\log}$-complete. Hence we can apply Corollary 10.1.6 and obtain $\leq_{\mathrm{m}}^{\log}$-mitoticity. □

We continue with classes $\mathcal{C}$ that are closed under $\leq_{1\text{-tt}}^{\log^2\text{-lin}}$-reducibility. For a set A that is $\leq_{2\text{-tt}}^{\log}$-complete for $\mathcal{C}$ we define a set D that simulates $\leq_{2\text{-tt}}^{\log}$-reductions to A. We define D in such a way that it can be reduced to A with one query of linear length in space $O(\log(n)^2)$, hence we will have $D \in \mathcal{C}$, so $D \leq_{2\text{-tt}}^{\log} A$ via some machine M_j. We will make sure that whenever M_j queries x, we have information about x by another query different from x. This will show that A is autoreducible.

Theorem 10.1.8 *If a class $\mathcal{C}$ is closed under $\leq_{1\text{-tt}}^{\log^2\text{-lin}}$-reducibility, then all $\leq_{2\text{-tt}}^{\log}$-complete sets for $\mathcal{C}$ are $\leq_{2\text{-tt}}^{\log}$-autoreducible.*

Proof Let $\{M_i\}_i$ be an enumeration of all $\leq_{2\text{-tt}}^{\log}$-reductions such that M_i on input x can be simulated in space $\log i \cdot (1 + \log |x|)$ using a binary alphabet. Without loss of generality, we assume that all reductions always query exactly two distinct values. Let A be $\leq_{2\text{-tt}}^{\log}$-complete for $\mathcal{C}$ and consider the algorithm described in Figure 10.2.

Let D denote the set decided by the above algorithm. The algorithm describes a $\leq_{1\text{-tt}}^{\log^2\text{-lin}}$-reduction to A, so we have $D \in \mathcal{C}$. This means that $D \leq_{2\text{-tt}}^{\log} A$ via some machine M_j.

For arbitrary x, let t denote the truth-table obtained in the algorithm for D on input $\langle 0^j, x\rangle$ with the query x replaced by $c_A(x)$ if possible. If t is constant, then we obtain that

$$\langle 0^j, x\rangle \in D \iff \text{the algorithm for } D \text{ accepts } \langle 0^j, x\rangle \iff t \text{ is false} \iff \langle 0^j, x\rangle \notin D,$$

```
On input $z$:
 1. if $z$ is not of the form $\langle 0^i, x\rangle$, then reject
 2. compute $c_A(x)$ with one query to the oracle $A$
 3. simulate $M_i$ on input $\langle 0^i, x\rangle$ and let $t$ be the truth-table that is obtained
    if we replace the query $x$ with $c_A(x)$ (if applicable)
 4. if $t$ is constant:
 5.   accept if $t$ is false, and reject if $t$ is true
 6. if $t$ depends negatively on one query:
 7.   accept if $c_A(x) = 0$, and reject if $c_A(x) = 1$
 8. accept if $c_A(x) = 1$, and reject if $c_A(x) = 0$
```

Figure 10.2: Definition of the set D.

a contradiction. Hence for all x we obtain that even if we replace x by $c_A(x)$ in t, then t is not constant.

For our autoreduction on input x, we consider $M_j(\langle 0^j, x\rangle)$ and distinguish the following cases.

Case 1: $M_j(\langle 0^j, x\rangle)$ does not query x. We have the following subcases:

- t depends positively on one query $y \neq x$. Then we have $x \in A \iff c_A(x) = 1 \iff \langle 0^j, x\rangle \in D \iff t(c_A(y))$ is true $\iff c_A(y) = 1$.
- t depends negatively on one query $y \neq x$. Then we have $x \in A \iff c_A(x) = 1 \iff \langle 0^j, x\rangle \notin D \iff t(c_A(y))$ is false $\iff c_A(y) = 1$.
- t depends on two queries $y_1, y_2 \neq x$. Then we have $x \in A \iff c_A(x) = 1 \iff \langle 0^j, x\rangle \in D \iff t(c_A(y_1), c_A(y_2))$ is true.

Hence in this case we can either behave like $c_A(y)$ or like $t(c_A(y_1), c_A(y_2))$.

Case 2: $M_j(\langle 0^j, x\rangle)$ queries x. Then, t either depends positively or negatively on the second query $y \neq x$. The argumentation in the first case shows that $x \in A \iff y \in A$.

Note that j is constant, so the simulation of $M_j(\langle 0^j, x\rangle)$ can be done in logspace, hence the above case distinction describes a $\le^{\log}_{2\text{-tt}}$-autoreduction for A. □

Corollary 10.1.9 *All $\le^{\log}_{2\text{-tt}}$-complete sets for the following classes are $\le^{\log}_{2\text{-tt}}$-autoreducible:* $\mathrm{QP} = \mathrm{DTIME}(2^{\mathrm{polylog}(n)})$*,* PSPACE*,* EXP*,* REC*,* DSPACE(s) *and* NSPACE(s) *for all space-constructible $s \geq \log^2$.*

Proof The classes satisfy the requirements in Theorem 10.1.8. □

10.2 Complete Sets for PSPACE and EXP

We continue by showing that for some restricted polynomial-time truth-table reductions, complete sets for EXP are complete under length-increasing reductions, and the same holds considering logspace reductions for PSPACE and EXP. By carefully repeating the length-increasing reductions in such a way that we switch between stages defined by a separator set we obtain mitoticity for PSPACE and EXP.

Definition 10.2.1 *Given two sets A and B, we define $A \leq^{\mathrm{p}}_{\text{T-li}} B$ if there exists an oracle Turing machine M such that $A = L(M^B)$ and all queries made by $M^B(x)$ are of length strictly greater than $|x|$. The following reducibility notions are defined similarly: $\leq^{\mathrm{p}}_{\text{2-tt-li}}$, $\leq^{\mathrm{p}}_{k\text{-ctt-li}}$, $\leq^{\mathrm{p}}_{k\text{-dtt-li}}$, $\leq^{\mathrm{p}}_{\text{dtt-li}}$, $\leq^{\mathrm{p}}_{\text{ctt-li}}$, $\leq^{\mathrm{p}}_{\text{m-li}}$, $\leq^{\log}_{\text{T-li}}$, $\leq^{\log}_{\text{2-tt-li}}$, $\leq^{\log}_{k\text{-ctt-li}}$, $\leq^{\log}_{k\text{-dtt-li}}$, $\leq^{\log}_{\text{dtt-li}}$, $\leq^{\log}_{\text{ctt-li}}$, $\leq^{\log}_{\text{m-li}}$.*

Berman [Ber77] and Ganesan and Homer [GH92] show that all many-one complete sets for EXP are many-one length-increasing equivalent. In the following lemma, we generalize their approach to show that for certain polynomial-time reductions and logspace reductions, completeness for EXP and PSPACE implies completeness with respect to length-increasing reductions.

Lemma 10.2.2 *Let $k \geq 1$.*

1. *All $\leq$-complete sets for* EXP *are $\leq_{\text{-li}}$ equivalent, where $\leq$ can be each of the following reducibility notions: $\leq^{\mathrm{p}}_{k\text{-ctt}}, \leq^{\mathrm{p}}_{k\text{-dtt}}, \leq^{\mathrm{p}}_{\text{ctt}}, \leq^{\mathrm{p}}_{\text{dtt}}$.*
2. *All $\leq$-complete sets for* PSPACE *and* EXP *are $\leq_{\text{-li}}$ equivalent, where $\leq$ can be each of the following reducibility notions: $\leq^{\log}_{k\text{-ctt}}, \leq^{\log}_{k\text{-dtt}}, \leq^{\log}_{\text{ctt}}, \leq^{\log}_{\text{dtt}}$.*

Proof We begin with $\leq^{\mathrm{p}}_{k\text{-dtt}}$-complete sets for EXP. Let A be $\leq^{\mathrm{p}}_{k\text{-dtt}}$-complete for EXP, and let $C := K \times \Sigma^*$ for some $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete set K for EXP. Then, C is $\leq^{\mathrm{p}}_{\text{m-li}}$-complete for EXP. So for all $D \in$ EXP it holds that $D \leq^{\mathrm{p}}_{\text{m-li}} C$, and it suffices to show $C \leq^{\mathrm{p}}_{k\text{-dtt-li}} A$.

Let $f_1, f_2, \ldots$ denote an enumeration of all $\leq^{\mathrm{p}}_{k\text{-dtt}}$-reductions such that $f_i(x)$ can be computed in time $n^i + i$. We define a set B by the algorithm described in Figure 10.3.

```
On input $\langle 0^i, x\rangle$:
 1. compute $\langle y_1, \ldots, y_k\rangle := f_i(\langle 0^i, x\rangle)$
 2. if $|y_l| \leq |\langle 0^i, x\rangle|$ and $y_l \in A$ for some $l$, then reject
 3. accept if $x \in C$, and reject otherwise
```

Figure 10.3: Definition of the set B.

Observe that $B \in$ EXP, so $B \leq^{\mathrm{p}}_{k\text{-dtt}} A$ via some reduction $f_j \in$ FP. We consider f_j on input $\langle 0^j, x\rangle$ with $n = |\langle 0^j, x\rangle|$ and obtain the following case distinction.

Case 1: There exists some $y_l \in f_j(\langle 0^j, x\rangle)$ such that $|y_l| \leq n$ and $y_l \in A$. We obtain $0 = c_B(\langle 0^j, x\rangle) = \max\{c_A(y_i) \mid y_i \in f_j(\langle 0^j, x\rangle)\} = c_A(y_l) = 1$, a contradiction. Hence this case cannot occur.

Case 2: For all $y_l \in f_j(\langle 0^j, x\rangle)$, if $|y_l| \leq n$, then $y_l \notin A$. Then we obtain

$$\begin{aligned} c_C(x) = c_B(\langle 0^j, x\rangle) &= \max\{c_A(y_i) \mid y_i \in f_j(\langle 0^j, x\rangle)\} \\ &= \max\{c_A(y_i) \mid y_i \in f_j(\langle 0^j, x\rangle) \text{ and } |y_i| > n\} \end{aligned}$$

In particular note that $n \geq |x|$.

On input x, let $Q_x := \{y_i \in f_j(\langle 0^j, x\rangle) \mid |y_i| > |\langle 0^j, x\rangle|\}$. Since only the second case can occur, we have $c_C(x) = \max\{c_A(y_i) \mid y_i \in Q_x\}$, where $|y_i| > |\langle 0^j, x\rangle| \geq |x|$ for all $y_i \in Q_x$. Hence Q_x shows that $C \leq^{\mathrm{p}}_{k\text{-dtt-li}} A$. Moreover, the above argumentation also holds if we omit the boundary k, hence it also shows that all $\leq^{\mathrm{p}}_{\text{dtt}}$-complete sets for EXP are $\leq^{\mathrm{p}}_{\text{dtt-li}}$-equivalent.

In a similar way we show the lemma for the reducibility notions $\leq^{\mathrm{p}}_{k\text{-ctt}}$ and $\leq^{\mathrm{p}}_{\mathrm{ctt}}$. In these cases, we modify line 2 of the algorithm for B such that we accept if $y_l \notin A$. Hence, if there exists some $y_l \in f_j(\langle 0^j, x\rangle)$ such that $|y_l| \leq n$ and $y_l \notin A$, then we obtain the contradiction $1 = c_B(\langle 0^j, x\rangle) = \min\{c_A(y_i) \mid y_i \in f_j(\langle 0^j, x\rangle)\} = c_A(y_l) = 0$. So for all $y_l \in f_j(\langle 0^j, x\rangle)$, if $|y_l| \leq n$, then $y_l \in A$, and hence for $Q_x := \{y_i \in f_j(\langle 0^j, x\rangle) \mid |y_i| > |\langle 0^j, x\rangle|\}$ we obtain $c_C(x) = c_B(\langle 0^j, x\rangle) = \min\{c_A(y_i) \mid y_i \in f_j(\langle 0^j, x\rangle)\} = \min\{c_A(y_i) \mid y_i \in Q_x\}$. This completes the proof of the first item of the lemma.

We show the second item of the lemma using an enumeration $f_1, f_2, \ldots$ of logspace reductions such that f_i can be simulated in space $\log(i) \cdot (1 + \log(n))$ using a binary alphabet. □

Lemma 10.2.3 *All $\leq^{\mathrm{p}}_{2\text{-tt}}$-complete sets for* EXP *are $\leq^{\mathrm{p}}_{2\text{-tt-li}}$ equivalent, and all $\leq^{\log}_{2\text{-tt}}$-complete sets for* PSPACE *and* EXP *are $\leq^{\log}_{2\text{-tt-li}}$ equivalent.*

Proof Let A be $\leq^{\mathrm{p}}_{2\text{-tt}}$-complete for EXP, and let $C := K \times \Sigma^*$ for some $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete set K for EXP. Then, C is $\leq^{\mathrm{p}}_{\mathrm{m\text{-}li}}$-complete for EXP. So for all $D \in$ EXP it holds that $D \leq^{\mathrm{p}}_{\mathrm{m\text{-}li}} C$, and it suffices to show $C \leq^{\mathrm{p}}_{2\text{-tt-li}} A$.

Let $\{M_i\}_{i\geq 1}$ be an enumeration of all $\leq^{\mathrm{p}}_{2\text{-tt}}$ reductions such that the computation of M_i on x can be simulated in time $|x|^i + i$. Note that for all $L_1, L_2 \subseteq \Sigma^*$ it holds that $L_1 \leq^{\mathrm{p}}_{2\text{-tt}} L_2$ if and only if there exist polynomial-time computable functions $f\colon \Sigma^* \to (\Sigma^*)^2$ and $g\colon \Sigma^* \times \{0,1\}^2 \to \{0,1\}$ such that for all x, $f(x) = \langle q_1, q_2\rangle$ and $c_{L_1}(x) = g(x, c_{L_2}(q_1), c_{L_2}(q_2))$. Let f_i and g_i be the functions that correspond to the reduction M_i. We assume that if $f_i(x) = \langle q_1, q_2\rangle$, then $|q_1| \leq |q_2|$. This is not a restriction, because we can always switch q_1 with q_2 and modify g_i on x accordingly.

Let B be the set of inputs $\langle 0^i, x\rangle$ accepted by the algorithm described in Figure 10.4.

```
On input $\langle 0^i, x\rangle$:
 1. compute $f_i(\langle 0^i, x\rangle) = \langle q_1, q_2\rangle$
 2. if $|\langle 0^i, x\rangle| < |q_1| \leq |q_2|$ and $x \in C$, then accept
 3. if $|q_1| \leq |q_2| \leq |\langle 0^i, x\rangle|$ and $g_i(x, c_A(q_1), c_A(q_2)) = 0$, then accept
 4. if $|q_1| \leq |\langle 0^i, x\rangle| < |q_2|$, then consider the following cases:
     (a) if $g_i(x, c_A(q_1), 0) = g_i(x, c_A(q_1), 1) = 0$, then accept
     (b) if $g_i(x, c_A(q_1), b) = b$ for all $b$, and $x \in C$, then accept
     (c) if $g_i(x, c_A(q_1), b) \neq b$ for all $b$, and $x \notin C$, then accept
 5. reject
```

Figure 10.4: Definition of the set B.

Claim 10.2.4 $B \in$ EXP.

Proof We consider the above algorithm on input $\langle 0^i, x\rangle$ with $n = |\langle 0^i, x\rangle|$:

- M_i on input x can be simulated in time $|x|^i + i$, where $i \leq n$, hence computing $f_i(\langle 0^i, x\rangle)$ in step 1 and $g_i(x, \alpha, \beta)$ in step 3 and step 4 is possible in exponential time in n.
- $C \in$ EXP, so deciding $x \in C$ in step 2 and step 4 takes exponential time in n.
- $A \in$ EXP and in step 3 we have $|q_1| \leq |q_2| \leq n$. Hence computing $c_A(q_1)$ and $c_A(q_2)$ in step 3 takes time exponential in n.
- Analogously, computing $c_A(q_1)$ in step 4 takes time exponential in n.

□

Since A is $\leq^{\mathrm{p}}_{2\text{-tt}}$-complete for EXP, we have $B \leq^{\mathrm{p}}_{2\text{-tt}} A$ by some reduction M_j. Let $f_j(\langle 0^j, x\rangle) = \langle q_1, q_2\rangle$. We distinguish the following cases in the execution of the algorithm for B on input $\langle 0^j, x\rangle$.

Case 1: $|\langle 0^j, x\rangle| < |q_1| \le |q_2|$. Then, $c_C(x) = c_B(\langle 0^j, x\rangle) = g_j(x, c_A(q_1), c_A(q_2))$. Notice that in this case, both q_1 and q_2 are longer than x.

Case 2: $|q_1| \le |q_2| \le |\langle 0^j, x\rangle|$. Then, $g_j(x, c_A(q_1), c_A(q_2)) \neq c_B(\langle 0^j, x\rangle) = g_j(x, c_A(q_1), c_A(q_2))$, where the inequality holds by the definition of B, and the equality holds because M_j reduces B to A. This is a contradiction, hence this case cannot occur.

Case 3: $|q_1| \le |\langle 0^j, x\rangle| < |q_2|$. We have the following subcases.

- $g_j(x, c_A(q_1), 0) = g_j(x, c_A(q_1), 1)$. Again we obtain $g_j(x, c_A(q_1), c_A(q_2)) \neq c_B(\langle 0^j, x\rangle)$ and $c_B(\langle 0^j, x\rangle) = g_j(x, c_A(q_1), c_A(q_2))$, a contradiction. So this subcase cannot occur.
- $g_j(x, c_A(q_1), b) = b$ for all b. Then, $c_C(x) = c_B(\langle 0^j, x\rangle) = g_j(x, c_A(q_1), c_A(q_2)) = c_A(q_2)$.
- $g_j(x, c_A(q_1), b) \neq b$ for all b. So, $c_C(x) = 1 - c_B(\langle 0^j, x\rangle) = 1 - g_j(x, c_A(q_1), c_A(q_2)) = c_A(q_2)$.

So, in this case we have $c_C(x) = c_A(q_2)$ with $|x| < |\langle 0^j, x\rangle| < |q_2|$.

By the above case distinction, the algorithm described in Figure 10.5 is a $\le^{\mathrm{p}}_{\text{2-tt-li}}$ reduction from C to A. Note that j is constant, hence the algorithm can be executed in polynomial time.

```
On input $x$:
 1. compute $f_j(\langle 0^j, x\rangle) = \langle q_1, q_2\rangle$
 2. if $|\langle 0^j, x\rangle| < |q_1| \le |q_2|$ and $g_j(x, c_A(q_1), c_A(q_2)) = 1$, then accept
 3. if $|q_1| \le |\langle 0^j, x\rangle| < |q_2|$ and $c_A(q_2) = 1$, then accept
 4. reject
```

Figure 10.5: A $\le^{\mathrm{p}}_{\text{2-tt-li}}$ reduction from C to A.

The case where we consider $\le^{\log}_{\text{2-tt}}$-complete sets for PSPACE and EXP is shown analogously, where we use an enumeration $f_1, f_2, \ldots$ of logspace reductions such that f_i can be simulated in space $\log(i) \cdot (1 + \log(n))$ using a binary alphabet. This completes the proof of the lemma. □

Glaßer et al. [GOP+07] show that if L is complete for some class with respect to many-one reductions that do not decrease the length too much, then L is many-one mitotic.

Theorem 10.2.5 ([GOP+07]) *Let $\mathcal{C}$ be a complexity class closed under many-one reductions. If L is many-one complete for $\mathcal{C}$ with respect to honest reductions, then L is many-one mitotic.*

We adapt their proof and obtain the following lemma.

Lemma 10.2.6 *Let A be a non-trivial set and let $\le$ be any of the following reductions with $k \ge 1$: $\le^{\mathrm{p}}_{\text{2-tt}}, \le^{\mathrm{p}}_{k\text{-ctt}}, \le^{\mathrm{p}}_{k\text{-dtt}}, \le^{\mathrm{p}}_{\text{ctt}}, \le^{\mathrm{p}}_{\text{dtt}}, \le^{\log}_{\text{2-tt}}, \le^{\log}_{k\text{-ctt}}, \le^{\log}_{k\text{-dtt}}, \le^{\log}_{\text{ctt}}, \le^{\log}_{\text{dtt}}$. If $(A \times \Sigma^*) \le_{\text{-}li} A$, then A is $\le$-mitotic.*

Proof We show the lemma for the case $\le^{\mathrm{p}}_{\text{ctt}}$, the other cases are shown analogously.

Choose $f \in \mathrm{FP}$ such that for all x there exists some m with $f(x) = \langle y_1, \ldots, y_m\rangle$, $|y_j| > |x|$ for all j, and $c_{(A\times\Sigma^*)}(x) = \min\{c_A(y_1), \ldots, c_A(y_m)\}$. Note that there is an $l > 0$ such that for all j it holds that $|x|^{1/l} < |y_j| < |x|^l$.

Define $t\colon \mathbb{N} \to \mathbb{N}$ by

$$t(i) = \begin{cases} 0 & \text{if } i = 0, \\ 2 & \text{if } i = 1, \text{ and} \\ t(i-1)^{2l^2} & \text{if } i > 1, \end{cases}$$

and let $S = \{x \mid \text{ for some odd } i \text{ it holds that } t(i) \leq |x| < t(i+1)\}$. Note that $S \in \mathrm{P}$.

Claim 10.2.7 *There exists a function $h \in \mathrm{FP}$ such that for all x there exists some m such that it holds that $h(x) = \langle y_1, \ldots, y_m \rangle$, $c_A(x) = \min\{c_A(y_1), \ldots, c_A(y_m)\}$, and $c_S(x) \neq c_S(y_j)$ for all j.*

Proof On input x, we first compute $i \in \mathbb{N}$ such that $t(i) \leq |x| < t(i+1)$. Observe that this can be done in polynomial time. Next we choose y large enough such that $t(i+1)^l < |\langle x, y \rangle| < t(i+1)^{2l}$. This is possible in polynomial time, because $t(i+1)^l = (t(i)^{2l^2})^l = t(i)^{2l^3} \leq |x|^{2l^3}$, where l is constant. Let $\langle y_1, \ldots, y_m \rangle = f(\langle x, y \rangle)$. Then, for each y_j it holds that

$$t(i+1) = (t(i+1)^l)^{1/l} < |\langle x, y \rangle|^{1/l} < |y_j| < |\langle x, y \rangle|^l < (t(i+1)^{2l})^l = t(i+1)^{2l^2} = t(i+2)$$

and hence we have $t(i) \leq |x| < t(i+1) < |y_j| < t(i+2)$, which means that $c_S(x) \neq c_S(y_j)$ for all j. Moreover, we have $c_A(x) = c_{(A \times \Sigma^*)}(\langle x, y \rangle) = \min\{c_A(y_1), \ldots, c_A(y_m)\}$, hence $h(x) = \langle y_1, \ldots, y_m \rangle$ proves the claim. $\square$

We next show that $A \equiv^{\mathrm{p}}_{\mathrm{ctt}} A \cap S$ and $A \cap S \leq^{\mathrm{p}}_{\mathrm{ctt}} A \cap \overline{S}$.

- $A \cap S \leq^{\mathrm{p}}_{\mathrm{ctt}} A$: We first compute $c_S(x)$ in polynomial time. If $c_S(x) = 1$, then we map to x. Otherwise we have $x \notin A \cap S$, and we map to some fixed element outside A.
- $A \leq^{\mathrm{p}}_{\mathrm{ctt}} A \cap S$: We first compute $c_S(x)$ in polynomial time. If $c_S(x) = 1$, then we map to x. Otherwise we have $c_S(x) = 0$. Let $\langle y_1, \ldots, y_m \rangle = h(x)$. By Claim 10.2.7 it holds that $c_S(y_j) = 1$ for all j, which means that $c_{(A \cap S)}(y_j) = c_A(y_j)$. By Claim 10.2.7 we further know that $c_A(x) = \min\{c_A(y_1), \ldots, c_A(y_m)\} = \min\{c_{(A \cap S)}(y_1), \ldots, c_{(A \cap S)}(y_m)\}$, which shows that $A \leq^{\mathrm{p}}_{\mathrm{ctt}} A \cap S$.
- $A \cap S \leq^{\mathrm{p}}_{\mathrm{ctt}} A \cap \overline{S}$: We first compute $c_S(x)$ in polynomial time. If $c_S(x) = 0$, then we have $x \notin A \cap S$, and we map to some fixed element outside A. Otherwise we have $c_S(x) = 1$. Let $\langle y_1, \ldots, y_m \rangle = h(x)$. By Claim 10.2.7 it holds that $c_S(y_j) = 0$ for all j, which means that $c_{(A \cap \overline{S})}(y_j) = c_A(y_j)$. By Claim 10.2.7 we further know that $c_{(A \cap S)}(x) = c_A(x) = \min\{c_A(y_1), \ldots, c_A(y_m)\} = \min\{c_{(A \cap \overline{S})}(y_1), \ldots, c_{(A \cap \overline{S})}(y_m)\}$, which shows that $A \cap S \leq^{\mathrm{p}}_{\mathrm{ctt}} A \cap \overline{S}$.

The symmetric cases $A \equiv^{\mathrm{p}}_{\mathrm{ctt}} A \cap \overline{S}$ and $A \cap \overline{S} \leq^{\mathrm{p}}_{\mathrm{ctt}} A \cap S$ are shown analogously. Hence A, $A \cap S$, and $A \cap \overline{S}$ are mutually $\leq^{\mathrm{p}}_{\mathrm{ctt}}$-equivalent for $S \in \mathrm{P}$, which shows that A is $\leq^{\mathrm{p}}_{\mathrm{ctt}}$-mitotic. $\square$

Corollary 10.2.8 *Let $k \geq 1$.*

1. *For* EXP, *all $\leq$-complete sets are $\leq$-mitotic, where $\leq$ can be each of the following reducibility notions: $\leq^{\mathrm{p}}_{2\text{-tt}}$, $\leq^{\mathrm{p}}_{k\text{-ctt}}$, $\leq^{\mathrm{p}}_{k\text{-dtt}}$, $\leq^{\mathrm{p}}_{\mathrm{ctt}}$, $\leq^{\mathrm{p}}_{\mathrm{dtt}}$.*
2. *For* PSPACE *and* EXP, *all $\leq$-complete sets are $\leq$-mitotic, where $\leq$ can be each of the following reducibility notions: $\leq^{\log}_{2\text{-tt}}$, $\leq^{\log}_{k\text{-ctt}}$, $\leq^{\log}_{k\text{-dtt}}$, $\leq^{\log}_{\mathrm{ctt}}$, $\leq^{\log}_{\mathrm{dtt}}$.*

Proof For the class $\mathcal{C}$ and the reducibility $\leq$, let A be $\leq$-complete for $\mathcal{C}$. Then, $A \times \Sigma^*$ is also $\leq$-complete for $\mathcal{C}$. By Lemma 10.2.2 and Lemma 10.2.3 we have $(A \times \Sigma^*) \leq_{\text{-li}} A$. By Lemma 10.2.6 it holds that A is $\leq$-mitotic. $\square$

10.3 Complete Sets for NEXP

The results in the previous two sections rely on the fact that the classes we considered are closed under complementation. For NEXP, this property is not known, so we cannot directly apply our results to NEXP. Instead we will show that complete sets for NEXP are autoreducible.

Theorem 10.3.1 ([NS12, GNR$^+$13a]) *Let $k \geq 1$. For* NEXP*, all $\leq$-complete sets are $\leq$-autoreducible, where $\leq$ can be each of the following reducibility notions: $\leq^{\mathrm{p}}_{k\text{-dtt}}$, $\leq^{\mathrm{p}}_{k\text{-ctt}}$, $\leq^{\mathrm{p}}_{\mathrm{dtt}}$, $\leq^{\mathrm{p}}_{\mathrm{ctt}}$, $\leq^{\log}_{\mathrm{m}}$, $\leq^{\log}_{k\text{-dtt}}$, $\leq^{\log}_{k\text{-ctt}}$, $\leq^{\log}_{\mathrm{dtt}}$, $\leq^{\log}_{\mathrm{ctt}}$.*

Proof Let A be a $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-complete set for NEXP. Hence there exists a nondeterministic Turing machine M that accepts A and runs in nondeterministic exponential time. Recall that $A \leq^{\mathrm{p}}_{\mathrm{dtt}} B$ if and only if there exists a polynomial-time computable function f such that for all x, $f(x) = \langle q_1, \dots, q_n\rangle$ and $c_A(x) = \max\{c_B(q_1), \dots, c_B(q_n)\}$. Let $\{f_i\}_{i\geq 1}$ be an enumeration of all polynomial-time Turing transducers such that the computation of f_i on x can be simulated in time $|x|^i + i$. Let B be the set of inputs $\langle 0^i, x\rangle$ that are accepted by the algorithm described in Figure 10.6.

```
On input $\langle 0^i, x\rangle$:
  1. $Q :=$ set of all queries of $f_i$ on input $\langle 0^i, x\rangle$
  2. if $x \notin Q$, then accept if there exists an accepting path in $M(x)$
  3. reject
```

Figure 10.6: Definition of the set B.

On input $\langle 0^i, x\rangle$ with $n = |\langle 0^i, x\rangle|$, we first simulate f_i on $\langle 0^i, x\rangle$. This takes time $n^i + i \leq n^n + n$, which is exponential in n. Then we compute $c_Q(x)$, which again takes exponential time, and either behave like $M(x)$ or reject. Since M is a nondeterministic exponential time machine we obtain $B \in$ NEXP. So $B \leq^{\mathrm{p}}_{\mathrm{dtt}} A$ via some disjunctive truth-table reduction f_j.

Let x be some input. We distinguish the following two cases.

Case 1: The reduction $f_j(\langle 0^j, x\rangle)$ queries x. Then, by the above algorithm, $\langle 0^j, x\rangle \notin B$. Hence for each query q of $f_j(\langle 0^j, x\rangle)$ we have $q \notin A$, and in particular it holds that $x \notin A$.

Case 2: The reduction $f_j(\langle 0^j, x\rangle)$ does not query x. Let $\langle q_1, \dots, q_m\rangle = f_j(\langle 0^j, x\rangle)$. Hence it holds that $x \neq q_i$ for all i. In this case, in the algorithm for B we behave like M on input x. Hence we obtain $c_A(x) = c_B(\langle 0^j, x\rangle) = \max\{c_A(q_1), \dots, c_A(q_m)\}$.

Based on this observation, we obtain the $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-autoreduction for A as described in Figure 10.7.

```
On input $x$:
  1. $Q :=$ set of all queries of $f_j$ on input $\langle 0^j, x\rangle$
  2. if $x \notin Q$, then return $Q$
  3. return some fixed value $x_0 \in (\overline{A} - \{x\})$
```

Figure 10.7: A $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-autoreduction for A.

The case of $\leq^{\mathrm{p}}_{\mathrm{ctt}}$-complete sets can be shown analogously. In this case, in line 3 of the algorithm for B we accept. Then, if $f_j(\langle 0^j, x\rangle)$ queries x, it holds that $\langle 0^j, x\rangle \in B$, hence all queries and in particular x are contained in A. Otherwise x is not queried, and $c_A(x) =$

$c_B(\langle 0^j, x\rangle) = \min\{c_A(q_1), \dots, c_A(q_m)\}$. So we obtain an $\leq^{\mathrm{p}}_{\mathrm{ctt}}$-autoreduction for A from the $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-autoreduction by returning some fixed value $x_1 \in (A - \{x\})$ in line 3.

The case of $\leq^{\mathrm{p}}_{k\text{-ctt}}$-complete sets and $\leq^{\mathrm{p}}_{k\text{-dtt}}$-complete sets follows from the above proof, if we replace the enumeration of all f_i by an enumeration of those f_i whose outputs are bounded by k, because in the autoreduction we return either a single query or we return the set $Q = f_j(\langle 0^j, x\rangle)$.

Finally, the case of logspace reducibilities can be obtained analogously by considering an enumeration where f_i can be simulated in space $\log(i) \cdot (1 + \log(n))$ using a binary alphabet. The case of $\leq^{\log}_{\mathrm{m}}$-complete sets follows from the case of $\leq^{\log}_{k\text{-dtt}}$-complete sets for $k = 1$. □

Corollary 10.3.2 *All $\leq^{\log}_{1\text{-tt}}$-complete sets for* NEXP *are $\leq^{\log}_{\mathrm{m}}$-autoreducible.*

Proof Buhrman [Buh93] showed that all $\leq^{\mathrm{p}}_{1\text{-tt}}$-complete sets for NEXP are $\leq^{\mathrm{p}}_{\mathrm{m}}$-complete. His proof also shows that all $\leq^{\log}_{1\text{-tt}}$-complete sets for NEXP are $\leq^{\log}_{\mathrm{m}}$-complete. We apply Theorem 10.3.1 and obtain $\leq^{\log}_{\mathrm{m}}$-autoreducibility. □

Theorem 10.3.3 ([NS12, GNR$^+$13a]) *Every $\leq$-complete set for* NEXP *is $\leq$-autoreducible, where $\leq$ can be each of the following reducibility notions: $\leq^{\log}_{2\text{-tt}}$, $\leq^{\mathrm{p}}_{2\text{-tt}}$.*

Proof We first show that $\leq^{\mathrm{p}}_{2\text{-tt}}$-complete sets for NEXP are $\leq^{\mathrm{p}}_{2\text{-tt}}$-autoreducible.

Let A be a $\leq^{\mathrm{p}}_{2\text{-tt}}$-complete set for NEXP. Hence there exists a nondeterministic Turing machine M that accepts A and runs in nondeterministic exponential time. Let $\{M_i\}_{i \geq 1}$ be an enumeration of all $\leq^{\mathrm{p}}_{2\text{-tt}}$ reductions such that the computation of M_i on x can be simulated in time $|x|^i + i$. Observe that $L_1 \leq^{\mathrm{p}}_{2\text{-tt}} L_2$ if and only if there exist polynomial-time computable functions $f\colon \Sigma^* \to (\Sigma^*)^2$ and $g\colon \Sigma^* \times \{0,1\}^2 \to \{0,1\}$ such that for all x it holds that $f(x) = \langle q_1, q_2\rangle$ and $c_{L_1}(x) = g(x, c_{L_2}(q_1), c_{L_2}(q_2))$. For each i, let f_i and g_i be the functions that correspond to the reduction M_i. Without loss of generality we can take the following assumptions:

- if $f_i(x) = \langle q_1, q_2\rangle$, then $q_1 \neq q_2$
- if $f_i(\langle 0^i, x\rangle) = \langle q_1, q_2\rangle$ and $x \in \{q_1, q_2\}$, then $x = q_1$

This is not a restriction, because if $q_1 = q_2$, then we can change the second query to a value different from q_1 and change g_i such that it ignores the answer to the second query, and if $x = q_2$ then we can switch q_1 and q_2 and again modify g_i accordingly. For each i and x, we further define the binary Boolean function $g_i^x\colon \{0,1\}^2 \to \{0,1\}$ by $g_i^x(\alpha, \beta) = g_i(x, \alpha, \beta)$.

Let $f_0^2, f_1^2, \dots, f_{15}^2$ denote an enumeration of all binary Boolean functions such that

$$i = 2^3 \cdot f_i^2(1,1) + 2^2 \cdot f_i^2(1,0) + 2^1 \cdot f_i^2(0,1) + 2^0 \cdot f_i^2(0,0)$$

holds for all $0 \leq i \leq 15$. So, for instance, f_{15}^2 denotes the binary constant 1, while f_8^2 denotes the binary *and*, and f_{14}^2 denotes the binary *or*.

Let B be the set of inputs $\langle 0^i, x\rangle$ that are accepted by the algorithm described in Figure 10.8.

Observe that $B \in$ NEXP. So $B \leq^{\mathrm{p}}_{2\text{-tt}} A$ via some $\leq^{\mathrm{p}}_{2\text{-tt}}$ reduction M_j. We consider the algorithm for B on some input $\langle 0^j, x\rangle$. Let $f_j(\langle 0^j, x\rangle) = \langle q_1, q_2\rangle$ and $Q = \{q_1, q_2\}$. If $x \notin Q$, we either accept in line 2 or we reject in line 4, depending on the behavior of $M(x)$. Hence it holds that $c_A(x) = c_B(\langle 0^j, x\rangle) = g_j(x, c_A(q_1), c_A(q_2))$ and $x \notin \{q_1, q_2\}$.

So suppose $x \in Q$. By our assumption on f_j we have $x = q_1$ and $x \neq q_2$. We have the following cases.

```
On input $\langle 0^i, x\rangle$:
 1. compute $f_i(\langle 0^i, x\rangle) = \langle q_1, q_2\rangle$ and let $Q = \{q_1, q_2\}$
 2. if $x \notin Q$ and there exists an accepting path in $M(x)$, then accept
 3. if $x \in Q$, then consider the following cases:
    (a) if $g_i^x \in \{f_0^2, f_1^2, f_2^2, f_4^2, f_8^2, f_9^2\}$, then accept
    (b) if $g_i^x \in \{f_5^2, f_{10}^2\}$ and there is an accepting path in $M(x)$, then accept
 4. reject
```

Figure 10.8: Definition of the set B.

Case 1: $g_j^x \in \{f_0^2, f_{15}^2\}$. If $g_j^x = f_0^2$, then $c_B(\langle 0^j, x\rangle) = 1$ and $g_j(x, c_A(x), c_A(q_2)) = 0$. If $g_j^x = f_{15}^2$, then $c_B(\langle 0^j, x\rangle) = 0$ and $g_j(x, c_A(x), c_A(q_2)) = 1$. In both subcases we obtain a contradiction to the fact that M_j reduces B to A. Hence this case cannot occur.

Case 2: $g_j^x \in \{f_1^2, f_2^2, f_{12}^2, f_{13}^2, f_{14}^2\}$.

- If $g_j^x = f_1^2$, then $1 = c_B(\langle 0^j, x\rangle) = f_1^2(c_A(x), c_A(q_2))$, hence $c_A(x) = 0$.
- If $g_j^x = f_2^2$, then $1 = c_B(\langle 0^j, x\rangle) = f_2^2(c_A(x), c_A(q_2))$, hence $c_A(x) = 0$.
- If $g_j^x = f_{12}^2$, then $0 = c_B(\langle 0^j, x\rangle) = f_{12}^2(c_A(x), c_A(q_2)) = c_A(x)$, hence $c_A(x) = 0$.
- If $g_j^x = f_{13}^2$, then $0 = c_B(\langle 0^j, x\rangle) = f_{13}^2(c_A(x), c_A(q_2))$, hence $c_A(x) = 0$.
- If $g_j^x = f_{14}^2$, then $0 = c_B(\langle 0^j, x\rangle) = f_{14}^2(c_A(x), c_A(q_2))$, hence $c_A(x) = 0$.

So in this case it holds that $c_A(x) = 0$.

Case 3: $g_j^x \in \{f_3^2, f_4^2, f_7^2, f_8^2, f_{11}^2\}$.

- If $g_j^x = f_3^2$, then $0 = c_B(\langle 0^j, x\rangle) = f_3^2(c_A(x), c_A(q_2)) = 1 - c_A(x)$, hence $c_A(x) = 1$.
- If $g_j^x = f_4^2$, then $1 = c_B(\langle 0^j, x\rangle) = f_4^2(c_A(x), c_A(q_2))$, hence $c_A(x) = 1$.
- If $g_j^x = f_7^2$, then $0 = c_B(\langle 0^j, x\rangle) = f_7^2(c_A(x), c_A(q_2))$, hence $c_A(x) = 1$.
- If $g_j^x = f_8^2$, then $1 = c_B(\langle 0^j, x\rangle) = f_8^2(c_A(x), c_A(q_2))$, hence $c_A(x) = 1$.
- If $g_j^x = f_{11}^2$, then $0 = c_B(\langle 0^j, x\rangle) = f_{11}^2(c_A(x), c_A(q_2))$, hence $c_A(x) = 1$.

So in this case it holds that $c_A(x) = 1$.

Case 4: $g_j^x \in \{f_5^2\}$. Then, $c_A(x) = c_B(\langle 0^j, x\rangle) = f_5^2(c_A(x), c_A(q_2)) = 1 - c_A(q_2)$. So in this case it holds that $c_A(x) = 1 - c_A(q_2)$.

Case 5: $g_j^x \in \{f_6^2, f_9^2, f_{10}^2\}$.

- If $g_j^x = f_6^2$, then $0 = c_B(\langle 0^j, x\rangle) = f_6^2(c_A(x), c_A(q_2))$, hence $c_A(x) = c_A(q_2)$.
- If $g_j^x = f_9^2$, then $1 = c_B(\langle 0^j, x\rangle) = f_9^2(c_A(x), c_A(q_2))$, hence $c_A(x) = c_A(q_2)$.
- If $g_j^x = f_{10}^2$, then $c_A(x) = c_B(\langle 0^j, x\rangle) = f_{10}^2(c_A(x), c_A(q_2)) = c_A(q_2)$.

So in this case it holds that $c_A(x) = c_A(q_2)$.

With this case distinction in mind, we can also handle the case where $x \in Q$ by either rejecting or accepting directly, or mapping to q_2 accordingly. We obtain a $\leq_{2\text{-tt}}^{\mathrm{p}}$-autoreduction for A as described in Figure 10.9.

```
On input $x$:
 1. compute $\langle q_1, q_2\rangle = f_j(\langle 0^j, x\rangle)$ and let $Q = \{q_1, q_2\}$
 2. if $x \notin Q$ and $g_j(x, c_A(q_1), c_A(q_2)) = 1$, then accept
 3. if $x \in Q$, then consider the following cases:
    (a) if $g_j^x \in \{f_3^2, f_4^2, f_7^2, f_8^2, f_{11}^2\}$, then accept
    (b) if $g_j^x \in \{f_5^2\}$ and $c_A(q_2) = 0$, then accept
    (c) if $g_j^x \in \{f_6^2, f_9^2, f_{10}^2\}$ and $c_A(q_2) = 1$, then accept
 4. reject
```

Figure 10.9: A $\leq^{\mathrm{p}}_{\text{2-tt}}$-autoreduction for A.

In order to show the statement for $\leq^{\log}_{\text{2-tt}}$-complete sets, suppose that A is $\leq^{\log}_{\text{2-tt}}$-complete for NEXP. Observe that the above enumeration $\{M_i\}_{i\geq 1}$ includes all $\leq^{\log}_{\text{2-tt}}$-reductions. If $B \leq^{\log}_{\text{2-tt}} A$ via the $\leq^{\log}_{\text{2-tt}}$ reduction M_j, then f_j and g_j are logspace computable, and hence the described autoreduction of A on input x can be computed in logspace. □

10.4 Summary and Discussion

We demonstrated that diagonalization is a powerful technique to establish redundancy properties of complete sets. We considered classes such as PSPACE and EXP and enforced logspace complete sets to be complete with respect to length-increasing reductions. This already implied autoreducibility, and by standard techniques we were able to turn this into mitoticity. Similar techniques established mitoticity for polynomial-time complete sets for EXP.

Our mitoticity results relied upon closure under complementation. Consequently, similar techniques applied to NEXP only showed autoreducibility. Recent work by Nguyen and Selman [NS14] studies negative results for complete sets for NEXP. The question remains open whether logspace complete sets for NEXP with respect to different reducibility notions are mitotic.

Chapter 11

Deterministic Coin Tossing

Autoreducibility versus Mitoticity We show that in some cases, autoreducibility of complete sets implies weak mitoticity. Recall that in general, mitoticity implies autoreducibility. Depending on the reducibility notion, the converse direction does not always hold. For instance, Ambos-Spies [AS84] showed that $\leq_{\mathrm{T}}^{\mathrm{p}}$-autoreducibility does not imply $\leq_{\mathrm{T}}^{\mathrm{p}}$-mitoticity. Glaßer et al. [GPSZ08] showed that for non-trivial sets, $\leq_{\mathrm{m}}^{\mathrm{p}}$-autoreducibility implies $\leq_{\mathrm{m}}^{\mathrm{p}}$-mitoticity, hence the concepts of $\leq_{\mathrm{m}}^{\mathrm{p}}$-autoreducibility and $\leq_{\mathrm{m}}^{\mathrm{p}}$-mitoticity are equivalent. They also showed that the same holds for $\leq_{\text{1-tt}}^{\mathrm{p}}$-reductions, but not for reductions between $\leq_{\text{3-tt}}^{\mathrm{p}}$ and $\leq_{\mathrm{T}}^{\mathrm{p}}$. We work on similar questions in the logspace setting. Glaßer [Gla10] showed that poly-logspace many-one autoreducibility implies poly-logspace many-one mitoticity. Moreover, he provided an oracle relative to which there exists a set L that is $\leq_{\mathrm{m}}^{\log}$-autoreducible and not $\leq_{\mathrm{m}}^{\log}$-mitotic. Hence, it is difficult to show that $\leq_{\mathrm{m}}^{\log}$-autoreducibility implies $\leq_{\mathrm{m}}^{\log}$-mitoticity. We will concentrate on weak mitoticity instead.

Mitoticity by Ruling Sets We sketch our approach as follows. Suppose we have a set A with a many-one autoreduction function f, and we want to show that A is many-one mitotic. Given some input x, we know that the repeated application of f results in elements y with the same membership to A as x. We call these elements the *trace of f on x*. One approach to show mitoticity is to construct an *r-ruling set S for f* of low complexity. Such a set S has the property that after at most r applications of f, the membership to S changes. Now we can reduce $A \cap S$ to $A \cap \overline{S}$ by applying f at most r times on x to obtain an equivalent element in $A \cap \overline{S}$. Since S has low complexity, we can find out if we already changed the membership to S, or if we need to apply f another time. The reductions $A \cap \overline{S}$ to A and A to $A \cap S$ can be done in a similar way, and so we obtain mitoticity. For polynomial-time many-one autoreduction functions, Glaßer et al. [GPSZ08] showed how to construct such a ruling set that is polynomial-time decidable, hence $\leq_{\mathrm{m}}^{\mathrm{p}}$-autoreducibility implies $\leq_{\mathrm{m}}^{\mathrm{p}}$-mitoticity. Their construction of a ruling set S uses an idea that was originally developed by Cole and Vishkin [CV86] and called *deterministic coin tossing*. Glaßer [Gla10] further analyzed whether this approach is applicable in the logspace setting. For a logspace many-one autoreduction f for A, he showed how to define a logspace decidable set S such that after at most $O(\log\log)$ steps of f, the membership to S changes. His approach shows that poly-logspace many-one autoreducible sets are poly-logspace many-one mitotic, but it is difficult to find a logspace decidable ruling set that changes membership after at most a constant number of steps of f.

Weak Logspace Mitoticity Again consider a $\leq_{\mathrm{m}}^{\log}$-autoreduction f for some set A. Glaßer [Gla10] notes that the number of steps of f that are necessary to obtain a change in the membership of S can be dropped down to $O(\log^{(c)})$ for arbitrary c. We show that we can modify this result by shifting complexity from the reduction into the set S. We obtain a separator S' with complexity slightly above L such that after at most two steps we have a membership change of f. So, for instance, if we want to show $A \leq^{\mathrm{log}}_{\mathrm{2\text{-}dtt}} A \cap S'$ and $A \leq^{\mathrm{log}}_{\mathrm{2\text{-}dtt}} A \cap \overline{S'}$, for every input it suffices to consider the next two steps of f. If A is $\leq^{\mathrm{log}}_{\mathrm{m}}$-complete for some class $\mathcal{C}$ that contains S' and that is closed under intersection, then we know that $A \cap S'$ and $A \cap \overline{S'}$ are $\leq^{\mathrm{log}}_{\mathrm{2\text{-}dtt}}$-complete for $\mathcal{C}$, and hence we know that A is weakly $\leq^{\mathrm{log}}_{\mathrm{2\text{-}dtt}}$-mitotic. Note that the complexity of the separator set is only slightly above L, so we almost obtain $\leq^{\mathrm{log}}_{\mathrm{2\text{-}dtt}}$-mitoticity. We further show how to adapt our technique to similar reducibility notions.

Contributions We summarize our contributions in this chapter as follows.

1. *General weak logspace mitoticity results.* Let $\mathcal{C} \supseteq (\mathrm{DSPACE}(\log \cdot \log^{(c)}) \cap \mathrm{P})$ be some complexity class that is closed under intersection, for some $c > 0$. We show:
   (a) If A is $\leq^{\mathrm{log}}_{\mathrm{m}}$-complete for $\mathcal{C}$ and $\leq^{\mathrm{log}}_{\mathrm{m}}$-autoreducible, then A is weakly $\leq^{\mathrm{log}}_{\mathrm{2\text{-}dtt}}$-mitotic.
   (b) If A is $\leq^{\mathrm{log}}_{\mathrm{1\text{-}tt}}$-complete for $\mathcal{C}$ and $\leq^{\mathrm{log}}_{\mathrm{1\text{-}tt}}$-autoreducible, then A is weakly $\leq^{\mathrm{log}}_{\mathrm{2\text{-}tt}}$-mitotic.
2. *Applications to* P*,* Δ^{p}_{k} *and* NEXP*.* We show that the autoreducibility results of $\leq^{\mathrm{log}}_{\mathrm{m}}$-complete sets imply weak mitoticity. We summarize these results in Table 11.1.
3. *Weak mitoticity for* PSPACE*.* Recall that all $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-complete sets for PSPACE are $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-autoreducible. We show that if we grant a separator set enough resources, then it can determine a path in the disjunctive autoreduction tree with the "correct" behavior, hence we can think of the $\leq^{\mathrm{p}}_{\mathrm{dtt}}$-autoreduction as a $\leq^{\mathrm{p}}_{\mathrm{m}}$-autoreduction, and by a similar technique we obtain weak mitoticity. We summarize our improved results in Table 11.2.

reduction	P	Δ^{p}_{k}	NEXP
$\leq^{\mathrm{log}}_{\mathrm{m}}$	$\mathrm{W}^{\mathrm{log}}_{\mathrm{2\text{-}tt}}$	$\mathrm{W}^{\mathrm{log}}_{\mathrm{2\text{-}tt}}$	$\mathrm{W}^{\mathrm{log}}_{\mathrm{2\text{-}dtt}}$
$\leq^{\mathrm{log}}_{\mathrm{1\text{-}tt}}$			$\mathrm{W}^{\mathrm{log}}_{\mathrm{2\text{-}dtt}}$

Table 11.1: Weak logspace mitoticity results shown in this chapter. An entry $\mathrm{W}^{\mathrm{log}}_{\mathrm{r}}$ in row $\leq$ and column $\mathcal{C}$ means that every $\leq$-complete set for $\mathcal{C}$ is weakly $\leq^{\mathrm{log}}_{\mathrm{r}}$-mitotic.

reduction	PSPACE
$\leq^{\mathrm{p}}_{k\text{-dtt}}$	$\mathrm{W}^{\mathrm{p}}_{l\text{-dtt}}$
$\leq^{\mathrm{p}}_{\mathrm{dtt}}$	$\mathrm{W}^{\mathrm{p}}_{\mathrm{dtt}}$

Table 11.2: Weak polynomial-time mitoticity results shown in this chapter. An entry $\mathrm{W}^{\mathrm{p}}_{\mathrm{r}}$ in row $\leq$ and column $\mathcal{C}$ means that every $\leq$-complete set for $\mathcal{C}$ is weakly $\leq^{\mathrm{p}}_{\mathrm{r}}$-mitotic. We have $k \geq 2$ and $l = k^3 + k^2 + k$.

Organization of this Chapter We show how to obtain ruling sets for autoreduction functions in Section 11.1. We apply these results in the remaining sections. In Section 11.2 we consider many-one autoreduction functions, and in Section 11.3 we generalize our approach to truth-table autoreduction functions that map to a single value. These two sections show results for logspace-computable functions. In Section 11.4 we show that a similar approach works for polynomial-time computable disjunctive truth-table autoreduction functions. We conclude this chapter with a summary and discussion in Section 11.5.

11.1 Autoreductions and Ruling Sets

In order to transform many-one autoreducibility into mitoticity, we consider the trace of words obtained by the repeated application of the autoreduction to some input x. The challenge is to define a set S of low complexity such that when we follow such a trace for r steps, then we visit at least one word in S and at least one word in $\overline{S}$. In the field of parallel algorithms, Cole and Vishkin [CV86] developed a technique called deterministic coin tossing. Their technique helps us to construct such a set S. In their terminology, the set S is called an r-ruling set.

Recall that for arbitrary functions f, we denote by $f^{(i)}$ the i-th iteration of f.

Definition 11.1.1 *For a total function f, a set S is called* r-ruling set for f *if and only if for every x there exists an $i \leq r$ such that $c_S(x) \neq c_S(f^{(i)}(x))$.*

In this section we show that we can obtain a 2-ruling set S from autoreduction functions, where S is only slightly more complex than f. Glaßer [Gla10] shows how to apply the coin tossing technique for an autoreducible set L to obtain an autoreduction function g for L of relatively low complexity and a 1-ruling set S for g.

Theorem 11.1.2 ([Gla10]) *Let $k \geq 1$ be an integer and let L be a non-trivial $\leq_{\mathrm{m}}^{\log^k}$-autoreducible set. Then there exist a total function $g \in \mathrm{FSPACE}((\log^{k^7} n) \cdot \log\log n)$ and a set $S \in \mathrm{DSPACE}(\log^{k^4} n)$ such that for all x,*

1. *$c_L(x) = c_L(g(x))$, and*
2. *$c_S(x) \neq c_S(g(x))$.*

In the proof of Theorem 11.1.2, the author considers the following distance function d, where

$$d(x,y) := \begin{cases} 0 & \text{if } x = y, \text{ and} \\ \mathrm{sgn}(y-x) \cdot \lfloor \log(2\mathrm{abs}(y-x)) \rfloor & \text{otherwise.} \end{cases}$$

The author uses the following lemma for integers.

Lemma 11.1.3 ([Gla10]) *If z_1, z_2, and z_3 are integers such that $d(z_1,z_2) = d(z_2,z_3) \neq 0$, then there exist $i,j \in [1,3]$ such that for $r := d(z_1,z_2)$,*

$$\lfloor z_i/2^{\mathrm{abs}(r)} \rfloor \text{ is even} \iff \lfloor z_j/2^{\mathrm{abs}(r)} \rfloor \text{ is odd.}$$

We will next show how to generalize Theorem 11.1.2 such that we find a change in the membership to the separator after very few steps of the autoreduction function. Our proof is based on the proof of Theorem 11.1.2 that can be found in the paper of Glaßer [Gla10].

Lemma 11.1.4 *Let f be a polynomially length-bounded function such that $f(x) \neq x$ for all x. For all $k \geq 1$ there exists a set S, a constant c_0, and a polynomial q such that:*

1. *For all x there exists some $i \leq c_0 \cdot (\log^{(k)}(|x|) + 1)$ such that $c_S(x) \neq c_S(f^{(i)}(x))$, and for all $j \leq i$ it holds that $|f^{(j)}(x)| \leq q(|x|)$.*
2. *If $s(n) \geq \log(n)$ and $f \in \mathrm{FSPACE}(s)$, then $S \in \mathrm{DSPACE}(s)$.*
3. *If $t(n) \geq n$ and $f \in \mathrm{FTIME}(t)$, then $S \in \mathrm{DTIME}(O(t \circ q))$.*

Proof Let f be a function and $p(n) = n^c + c$ be a polynomial such that $f(x) \neq x$ and $|f(x)| \leq p(|x|)$ for all x. We assume that $c \geq 2$ is large enough such that $\log^{(k)}(p^{(3)}(n)) \geq 3$ holds for all n. According to this length bound we now define the tower function

$$l(i) := \begin{cases} 2 & \text{if } i = 0, \text{ and} \\ p(p(l(i-1))) & \text{otherwise.} \end{cases}$$

Note that for the inverse tower function $l^{-1}(n) := \min\{i \mid l(i) \geq n\}$ and for all n,

$$l^{-1}(p(p(n))) = l^{-1}(n) + 1. \tag{11.1}$$

So from f's length bound we obtain for all x,

$$l^{-1}(|f(x)|) \leq l^{-1}(|x|) + 1 \quad \text{and} \quad l^{-1}(|f(f(x))|) \leq l^{-1}(|x|) + 1. \tag{11.2}$$

We partition the set of all words as follows.

$$\begin{aligned} S_0 &:= \{x \mid l^{-1}(|x|) \equiv 0 \mod 2\} \\ S_1 &:= \{x \mid l^{-1}(|x|) \equiv 1 \mod 2\} \end{aligned}$$

We can decide S_0 and S_1 as follows. On input x, compute and store $n = |x|$. Then, starting with $m = 2$, determine how often we need to apply $p^{(2)}$ to m until we obtain a value larger than n. For this we only need to store values less than or equal to n, which is possible in space $\log(|x|)$. Hence $S_0, S_1 \in \mathrm{L}$.

Recall the distance function d we defined above and note that $d \in \mathrm{FL}$. Furthermore, it holds that $d(x, y) = 0$ if and only if $x = y$. We will iteratively apply d to some value x and its successors in f. We define

$$d_i(x) := \begin{cases} d(d_{i-1}(x), d_{i-1}(f(x))) & \text{if } i > 0, \text{ and} \\ x & \text{if } i = 0, \end{cases}$$

for all x and all $i \geq 0$. Next, we define S to be the set decided by the algorithm described in Figure 11.1.

```
On input $x$:
 1. let $y := f^{(1)}(x)$ and $z := f^{(2)}(x)$
 2. if $|x| < |y|$ and $c_{S_0}(x) = c_{S_1}(y)$ then accept
 3. if $|y| < |z|$ and $c_{S_0}(y) = c_{S_1}(z)$ then reject
 4. if $d_{k+1}(x) > d_{k+1}(y)$ then accept
 5. if $d_{k+1}(x) < d_{k+1}(y)$ then reject
 6. for $j := k+1$ downto 1 do
 7.   if $d_j(x) \neq 0$, then accept if $\lfloor d_{j-1}(x)/2^{\mathrm{abs}(d_j(x))} \rfloor$ is even, otherwise reject
 8. reject
```

Figure 11.1: Algorithm for the set S.

Claim 11.1.5 *If $t \geq n$ and $f \in \mathrm{FTIME}(t)$, then $S \in \mathrm{DTIME}(O(t(q(n))))$ for a polynomial q.*

Proof Let $t(n) \geq n$ such that $f \in \mathrm{FTIME}(t)$. On input x, let $n = |x|$. We proceed by computing and storing $f^{(1)}(x), f^{(2)}(x), \ldots, f^{(k+2)}(x)$. Since k is constant, this takes time

$O((k+2)\cdot t(q'(n))) = O(t(q'(n)))$, where q' is a polynomial such that $q'(n)$ bounds the length of each $f^{(i)}(x)$ for $i \le k+2$.

Next, consider lines 2 and 3. Those lines can be executed in time $O(q''(n))$ for some polynomial q'', because x, y, z are stored on some tape, are polynomially long, and $S_0, S_1 \in \mathrm{L} \subseteq \mathrm{P}$.

Observe that for each $i \le k+1$, $d_i(x)$ (and $d_i(y)$) can be computed by iteratively applying $d \in \mathrm{FL} \subseteq \mathrm{FP}$ at most $(k+1)^2$ times. This means that lines 4 and 5 can be executed in time $O((k+1)^2 \cdot q'''(n)) = O(q'''(n))$ for some polynomial q'''. Moreover, the remaining loop is iterated at most $k+1$ times, and again, the values that are computed inside the loop can be computed in time $O(q'''(n))$, hence the entire loop can be executed in time $O((k+1)\cdot q'''(n)) = O(q'''(n))$.

Overall, the set S can be decided in time $O(t(q(n)))$ for some polynomial q. □

Claim 11.1.6 *If $s(n) \ge \log(n)$ and $f \in \mathrm{FSPACE}(s)$, then $S \in \mathrm{DSPACE}(s)$.*

Proof Let $s(n) \ge \log(n)$ such that $f \in \mathrm{FSPACE}(s)$. First, observe that for all $i \le k+2$ we have $f^{(i)}(x) \in \mathrm{FSPACE}(s)$, which is shown by induction on i using the facts that $s \ge \log$ and that f is polynomially length-bounded. By the polynomial length bound of $f^{(i)}$, and by $f^{(i)} \in \mathrm{FSPACE}(s)$ and $d \in \mathrm{FL}$, we also obtain $d_i \in \mathrm{FSPACE}(s)$ for all $i \le k+1$. Hence, all variables and functions can be computed in $\mathrm{FSPACE}(s)$. Also recall that $S_0, S_1 \in \mathrm{L}$, so we can decide $x, y, z \in S_0, S_1$ in $\mathrm{DSPACE}(s)$. □

By the above two claims, item 2 and item 3 of the lemma hold (for some polynomial q). We now continue to prove the first item as well. First, choose some constant c' that is large enough such that

$$2 \cdot \log^{(k-1)}(c^3 \cdot \log(n+3c)) \le c' \cdot (\log^{(k)}(n) + 1) \tag{11.3}$$

holds for all n. We further define h as the function computed by the algorithm described in Figure 11.2.

```
On input $x$:
 1. $n := |x|$, $m := 6c' \cdot \lceil \log^{(k)}(n) + 1 \rceil + k + 7$
 2. for $i := 1$ to $m$
 3.    // here $|f^{(i)}(x)| \le p^{(3)}(n)$
 4.    if $c_S(x) \ne c_S(f^{(i)}(x))$ then return $f^{(i)}(x)$
 5. // this line is never reached
```

Figure 11.2: Algorithm for the function h.

Claim 11.1.7 ([Gla10]) *In the algorithm for h, the invariant in line 3 holds.*

Proof Assume there exists an $i \in [1, m]$ such that the algorithm reaches the i-th iteration of the loop and there it holds that $|f^{(i)}(x)| > p^{(3)}(n)$. Choose the smallest such i and let $r = l^{-1}(n)$. Note that $i \ge 4$, since f's length is bounded by $p(n)$.

Observe that by (11.1), $l^{-1}(|f^{(i)}(x)|) \ge l^{-1}(p(p(n))) = r+1$. From (11.1) we also obtain that either

$$l^{-1}(p(n)) = r \quad \text{and} \quad l^{-1}(p(p(n))) = r+1 \tag{11.4}$$

or

$$l^{-1}(p(n)) = l^{-1}(p(p(n))) = r+1 \quad \text{and} \quad l^{-1}(p^{(3)}(n)) = r+2. \tag{11.5}$$

Recall that by (11.2), if $j_0 < j_2$ such that $l^{-1}(|f^{(j_0)}(x)|) = r$ and $l^{-1}(|f^{(j_2)}(x)|) = r+2$, then there exists $j_1 \in (j_0, j_2-2]$ such that $l^{-1}(|f^{(j_1)}(x)|) = r+1$ and $l^{-1}(|f^{(j_1+1)}(x)|) = r+1$. If (11.4) holds, then $l^{-1}(|f(x)|) \le r$ and so there exists $j \in [2,i]$ such that for $u = f^{(j-2)}(x)$, $v = f^{(j-1)}(x)$ and $w = f^{(j)}(x)$ it holds that $l^{-1}(|u|) = l^{-1}(|v|) = r$ and $l^{-1}(|w|) = r+1$. If (11.5) holds, then there exists $j \in [2,i]$ such that for $u = f^{(j-2)}(x)$, $v = f^{(j-1)}(x)$ and $w = f^{(j)}(x)$ it holds that $l^{-1}(|u|) = l^{-1}(|v|) = r+1$ and $l^{-1}(|w|) = r+2$. In both cases we have $c_{S_0}(u) = c_{S_0}(v)$ and $c_{S_0}(v) = c_{S_1}(w)$. If we consider the algorithm for S, then we see that $u \notin S$ and $v \in S$. Therefore, in the algorithm for h, the condition in line 4 is either satisfied for $j-2 < i$ or for $j-1 < i$. This contradicts our assumption that we reach the i-th iteration of the loop. □

Claim 11.1.8 *The algorithm for h never reaches line 5.*

Proof Assume that for some input x, the algorithm reaches line 5. Let $n = |x|$ and $m = 6c' \cdot \lceil \log^{(k)}(n) + 1 \rceil + k + 7$. Let $x_i = f^{(i)}(x)$ for $i \ge 0$. Hence, for all $i \in [1,m]$ it holds that $c_S(x) = c_S(x_i)$. Without loss of generality let us assume that $x_i \in S$ for all $i \in [0,m]$.

All remaining arguments refer to the algorithm for S. For $i \in [1,m]$ it holds that the algorithm on input x_i does not stop in line 2, since otherwise the algorithm on input x_{i-1} stops in line 3 which contradicts the assumption $x_{i-1} \in S$ (note that if the algorithm on input x_i stops in line 2, then by (11.2), on input x_{i-1} it cannot stop in line 2 as well.) So for all $i \in [1,m]$, the algorithm on input x_i reaches line 4.

For $i \ge 1$, let y_i and z_i denote the values of the program variables y and z when the algorithm for S works on input x_i. We show that there are not too many elements $i \in [1,m]$ such that the algorithm on input x_i stops in line 4.

By Claim 11.1.7, for $i \in [1,m]$, $|x_i| \le p^{(3)}(n)$. First, we inductively show that for all $j \in [0,k]$ and all $i \in [1, m-j-1]$ it holds that $\mathrm{abs}(d_{j+1}(x_i)) \le 2 \cdot \log^{(j)}(p^{(3)}(n))$.

- Let $j = 0$ and $i \in [1, m-1]$. For $\mathrm{abs}(d_{j+1}(x_i))$ we obtain:

$$\begin{aligned}\mathrm{abs}(d_1(x_i)) &= \mathrm{abs}(d(d_0(x_i), d_0(x_{i+1}))) = \lfloor \log(2 \cdot \mathrm{abs}(d_0(x_{i+1}) - d_0(x_i)) \rfloor \\ &\le \log(2 \cdot 2^{p^{(3)}(n)}) = 1 + p^{(3)}(n) \le 2 \cdot p^{(3)}(n) = 2 \cdot \log^{(0)}(p^{(3)}(n))\end{aligned}$$

- Let $0 < j \le k$ and suppose the inequality holds for $j-1$ and all $i \in [1, m-(j-1)-1] = [1, m-j]$. For $i \in [1, m-j-1]$ we obtain:

$$\begin{aligned}\mathrm{abs}(d_{j+1}(x_i)) &= \mathrm{abs}(d(d_j(x_i), d_j(x_{i+1}))) = \lfloor \log(2 \cdot \mathrm{abs}(d_j(x_{i+1}) - d_j(x_i)) \rfloor \\ &\le \log(2 \cdot (\mathrm{abs}(d_j(x_{i+1})) + \mathrm{abs}(d_j(x_i)))) \\ &\le \log(2 \cdot 2 \cdot (2 \cdot \log^{(j-1)}(p^{(3)}(n)))) \\ &\le 3 + \log(\log^{(j-1)}(p^{(3)}(n))) \\ &\le 2 \cdot \log^{(j)}(p^{(3)}(n))\end{aligned}$$

  Note that the last step holds because we have chosen c (and hence p) large enough such that it holds that $\log^{(k)}(p^{(3)}(n)) \ge 3$ for all n.

We obtain that for all $i \in [1, m-k-1]$ it holds that

$$\begin{aligned}\mathrm{abs}(d_{k+1}(x_i)) &\le 2 \cdot \log^{(k)}(p^{(3)}(n)) = 2 \cdot \log^{(k)}(((n^c + c)^c + c)^c + c) \\ &\le 2 \cdot \log^{(k-1)}(\log((n+3c)^{c^3})) = 2 \cdot \log^{(k-1)}(c^3 \cdot \log(n+3c)) \\ &\le c' \cdot (\log^{(k)}(n) + 1),\end{aligned}$$

where the last step holds because of inequation (11.3).

We now consider the sequence of $d_{k+1}(x_i)$ for $i \in [1, m-k-4]$. This sequence is not increasing, since otherwise we stop in line 5 which contradicts the assumption $x_i \in S$. We have seen that the values in this sequence are integers in $[-c' \cdot (\log^{(k)}(n)+1), c' \cdot (\log^{(k)}(n)+1)]$. So the number of positions where the sequence decreases is at most

$$2c' \cdot (\log^{(k)}(n)+1) \leq \frac{m-k-7}{3}.$$

This shows that the number of $i \in [1, m-k-4]$ such that the algorithm on input x_i stops in line 4 is at most $(m-k-7)/3 = (m-k-4)/3 - 1$. By a pigeon hole argument, there exists an $i \in [1, m-k-4]$ such that the algorithm reaches the loop in line 6 for the inputs x_i, x_{i+1}, and x_{i+2}. We finish the proof of the claim with the following case distinction:

Case 1: On some $u \in \{x_i, x_{i+1}, x_{i+2}\}$, the algorithm terminates after less than $k+1$ iterations. Choose $v \in \{x_i, x_{i+1}, x_{i+2}\}$ such that the algorithm on input v terminates after $k_0 < k+1$ iterations, and for each $w \in \{x_i, x_{i+1}, x_{i+2}\}$, the algorithm on input w does not terminate after less than k_0 iterations. Let j_0 denote the value of the variable j of the algorithm on input v in the iteration where the algorithm terminates. Hence we have $d_{j_0}(v) \neq 0$.

Next we will show that for all $w \in \{x_i, x_{i+1}, x_{i+2}\}$ we have $d_{j_0+1}(w) = 0$. If $j_0 = k+1$ we have $d_{j_0+1}(w) = d_{k+2}(w) = d(d_{k+1}(w), d_{k+1}(f(w))) = 0$, because for w we reach the loop and do not terminate in line 4 or line 5, which is only possible if $d_{k+1}(w) = d_{k+1}(f(w))$. If $j_0 < k+1$ we have $d_{j_0+1}(w) = 0$, because, by the choice of v, the algorithm on input w does not terminate after less than k_0 iterations. So for all $w \in \{x_i, x_{i+1}, x_{i+2}\}$ we have $d_{j_0+1}(w) = 0$.

This means $d_{j_0}(x_i) = d_{j_0}(x_{i+1}) = d_{j_0}(x_{i+2})$. Together with $d_{j_0}(v) \neq 0$ we obtain $d_{j_0}(x_i) = d_{j_0}(x_{i+1}) = d_{j_0}(x_{i+2}) \neq 0$, hence $d(d_{j_0-1}(x_i), d_{j_0-1}(x_{i+1})) = d(d_{j_0-1}(x_{i+1}), d_{j_0-1}(x_{i+2})) \neq 0$.

By Lemma 11.1.3 it follows that there are $i_1, i_2 \in [i, i+2]$ such that

$$\lfloor d_{j_0-1}(x_{i_1})/2^{\mathrm{abs}(d_{j_0}(x_{i_1}))} \rfloor \text{ is even} \iff \lfloor d_{j_0-1}(x_{i_2})/2^{\mathrm{abs}(d_{j_0}(x_{i_2}))} \rfloor \text{ is odd}.$$

Hence the algorithm accepts x_{i_1} if and only if it rejects x_{i_2}. This contradicts $x_{i_1}, x_{i_2} \in S$.

Case 2: On each $u \in \{x_i, x_{i+1}, x_{i+2}\}$, the algorithm reaches the $(k+1)$-st iteration. Recall that $k \geq 1$. Since the algorithm does not stop in the k-th iteration we have $d_2(x_i) = d_2(x_{i+1}) = d_2(x_{i+2}) = 0$, which means that $d_1(x_i) = d_1(x_{i+1}) = d_1(x_{i+2}) = d(x_i, x_{i+1})$. Since $f(x_i) \neq x_i$ we have $x_i \neq x_{i+1}$, hence $d_1(x_i) = d_1(x_{i+1}) = d_1(x_{i+2}) \neq 0$. By Lemma 11.1.3 it follows that there are $i_1, i_2 \in [i, i+2]$ where

$$\lfloor x_{i_1}/2^{\mathrm{abs}(d_1(x_{i_1}))} \rfloor \text{ is even} \iff \lfloor x_{i_2}/2^{\mathrm{abs}(d_1(x_{i_2}))} \rfloor \text{ is odd}.$$

Hence the algorithm accepts x_{i_1} if and only if it rejects x_{i_2}. This contradicts $x_{i_1}, x_{i_2} \in S$.

In each case we obtain a contradiction, so our assumption was wrong, and the claim holds. □

Given the last two claims, we can now finish our proof of item 1 as well. By Claim 11.1.8, there must exist some iteration with $i \leq m = 6c' \cdot \lceil \log^{(k)}(n)+1 \rceil + k + 7 \leq (12c' + k + 7) \cdot (\log^{(k)}(n)+1)$ where the algorithm for h on input x stops, hence $c_S(x) \neq c_S(f^{(i)}(x))$. By Claim 11.1.7 it holds that $|f^{(j)}(x)| \leq p^{(3)}(|x|)$ for all $j \leq i$. Hence also item 1 of the lemma holds. Finally, note that we have shown item 1 and item 3 for different polynomials. Clearly we can choose a single polynomial q for which both items hold. □

Our main result in this section is the following lemma. It shows that we can shift complexity from the reduction function to the separator such that after at most two steps we find a change in the membership to the separator.

Lemma 11.1.9 *Let f be a polynomially length-bounded function such that $f(x) \neq x$ for all x. For every $k \geq 1$ there exist a set S and a function g with the following properties:*

1. $c_S(g(x)) \neq c_S(f(g(x)))$
2. $g(x) \in \{x, f(x)\}$
3. *If $s(n) \geq \log(n)$ and $f \in \mathrm{FSPACE}(s)$, then $S \in \mathrm{DSPACE}(s \cdot \log^{(k)})$ and $g \in \mathrm{FSPACE}(s)$.*
4. *If $t(n) \geq n$ and $f \in \mathrm{FTIME}(t)$, then $S \in \mathrm{DTIME}(O(t \circ q))$ and $g \in \mathrm{FTIME}(O(t \circ q))$, where q is some polynomial.*

Proof Let f be a function and p be a polynomial with $f(x) \neq x$ and $|f(x)| \leq p(|x|)$ for all x. By Lemma 11.1.4 there exist a set S', a polynomial p', and some constant c such that for all x there exists some $i \leq c \cdot (1 + \log^{(k)}(|x|))$ with $c_{S'}(x) \neq c_{S'}(f^{(i)}(x))$ and $|f^{(j)}(x)| \leq p'(|x|)$ for all $j \leq i$. We define S to be the set decided by the algorithm described in Figure 11.3.

```
On input $x$:
 1. $y := x$, $i := 0$
 2. while ($y \notin S'$ or $f(y) \in S'$):
 3.    $y := f(y)$, $i := i + 1$
 4. accept if $i$ is even, and reject otherwise
```

Figure 11.3: Algorithm for the set S.

Furthermore, we define the function g by

$$g(x) := \begin{cases} f(x) & \text{if } x \in S' \text{ and } f(x) \notin S' \\ x & \text{otherwise} \end{cases}$$

for all x. Note that by this definition, item 2 of the lemma holds for g.

We will next show that item 1 of the lemma holds. Let x be some arbitrary input. Recall that after at most $O(\log^{(k)}(|x|))$ iterations of f, the membership to S' changes. Choose the minimal l such that $f^{(l)}(x) \in S'$ and $f^{(l+1)}(x) \notin S'$. Note that $l \in O(\log^{(k)}(|x|))$ and $|f^{(l')}(x)| \leq p'(p'(|x|))$ for all $l' \leq l$. By the choice of l, for all $l' < l$ it holds that $f^{(l')}(x) \notin S'$ or $f^{(l'+1)}(x) \in S'$. This means that the algorithm stops after exactly l iterations. We distinguish the following cases:

Case 1: $l > 0$. By the minimality of l we have $x = f^{(0)}(x) \notin S'$ or $f(x) = f^{(1)}(x) \in S'$, which means that $g(x) = x$. Moreover, for $l' = l - 1$ it holds that $f^{(l')}(f(x)) \in S'$ and $f^{(l'+1)}(f(x)) \notin S'$, and l' is minimal with this property. Hence, on input $f(x)$, the algorithm for S stops after $l - 1$ iterations, and we obtain $g(x) \in S \iff x \in S \iff l$ is even $\iff l - 1$ is odd $\iff f(x) \notin S \iff f(g(x)) \notin S$.

Case 2: $l = 0$. This means that $x \in S'$ and $f(x) \notin S'$, and hence $g(x) = f(x)$. So, for the smallest $l' \in \mathbb{N}$ with $f^{(l')}(f(x)) \in S'$ and $f^{(l'+1)}(f(x)) \notin S'$ it holds that $l' > 0$, and we obtain $g(x) \in S \iff f(x) \in S \iff f(f(x)) \notin S \iff f(g(x)) \notin S$.

In each case we obtain $c_S(g(x)) \neq c_S(f(g(x)))$, so item 1 of the lemma holds for g. It remains to argue for the runtime and the space requirements of the algorithm. We have the following claims.

Claim 11.1.10 *If $f \in \mathrm{FTIME}(t)$ for some t with $t(n) \geq n$, then $S \in \mathrm{DTIME}(O(t(q(n))))$ and $g \in \mathrm{FTIME}(O(t(q(n))))$, where q is some polynomial.*

Proof Suppose that $t(n) \geq n$ and $f \in \mathrm{FTIME}(t)$. By Lemma 11.1.4 we have $S' \in \mathrm{DTIME}(O(t(q'(n))))$ for some polynomial q'. Then, $g \in \mathrm{FTIME}(O(t(q''(n))))$ for some polynomial q''. Next, we consider the algorithm for S. There exists a constant c' such that we iterate at most

$$l \leq c \cdot (1 + \log^{(k)}(|x|)) + c \cdot (1 + \log^{(k)}(p'(p'(|x|)))) \leq c' \cdot (1 + \log^{(k)}(|x|))$$

times. In each iteration, we compute f and decide S' on inputs of length at most $p'(p'(|x|))$, which is possible in time $O(t(p'(p'(|x|))))$ and $O(t(q'(p'(p'(|x|)))))$, respectively. Hence there exists a polynomial q'' such that the entire loop can be executed in time $O(c' \cdot (1 + \log^{(k)}(n)) \cdot t(q''(n)))$, and thus there exists a polynomial q such that the entire algorithm can be executed in time $O(t(q(n)))$. □

Claim 11.1.11 *If $s(n) \geq \log(n)$ and $f \in \mathrm{FSPACE}(s)$, then $S \in \mathrm{DSPACE}(s \cdot \log^{(k)})$ and $g \in \mathrm{FSPACE}(s)$.*

Proof Suppose that $s(n) \geq \log(n)$ and $f \in \mathrm{FSPACE}(s)$. By Lemma 11.1.4 we have $S' \in \mathrm{DSPACE}(s)$, hence $g \in \mathrm{FSPACE}(s)$ (here we need to recompute each bit of the function f on input x, which is possible in space s since $|f(x)| \leq p(|x|)$ and $s(n) \geq \log(n)$).

Next, consider the algorithm for S. We have already argued that we iterate at most $c' \cdot (1 + \log^{(k)}(|x|))$ times, and that in each iteration, the length of y is bounded by $p'(p'(|x|))$, and hence in each iteration, the single bits of y can be addressed by $O(\log(|x|))$ many bits. So by a blockwise recomputation we obtain $S \in \mathrm{DSPACE}(O(s(n) \cdot c' \cdot (1 + \log^{(k)}(n)))) = \mathrm{DSPACE}(s(n) \cdot \log^{(k)}(n))$. This shows the claim. □

□

11.2 Many-One Complete Sets

Given a $\leq_{\mathrm{m}}^{\log}$-autoreduction f for some set A that is $\leq_{\mathrm{m}}^{\log}$-complete for some class $\mathcal{C}$, we apply the main result of the previous section to generate a 2-ruling set S for f. Since the complexity of S is only slightly higher than the complexity of f, we will obtain $A \cap S \in \mathcal{C}$ and $A \cap \overline{S} \in \mathcal{C}$. Considering some input x, we then find two elements y, z on f's trace on x with the same membership to A as x such that exactly one is contained in S. Hence, $\{y, z\}$ reduces A to $A \cap S$ and $A \cap \overline{S}$. So A is $\leq_{\mathrm{m}}^{\log}$-complete for $\mathcal{C}$, and $A \cap S$ and $A \cap \overline{S}$ are $\leq_{\text{2-dtt}}^{\log}$-complete for $\mathcal{C}$. This shows that A is weakly $\leq_{\text{2-dtt}}^{\log}$-mitotic.

Theorem 11.2.1 *Let $c \geq 1$ and let $\mathcal{C} \supseteq (\mathrm{DSPACE}(\log \cdot \log^{(c)}) \cap \mathrm{P})$ be closed under intersection. If A is $\leq_{\mathrm{m}}^{\log}$-complete for $\mathcal{C}$ and $\leq_{\mathrm{m}}^{\log}$-autoreducible, then A is weakly $\leq_{\text{2-dtt}}^{\log}$-mitotic.*

Proof Let $f \in \mathrm{FL}$ be a $\leq_{\mathrm{m}}^{\log}$-autoreduction for A. From Lemma 11.1.9 we obtain a set $S \in (\mathrm{DSPACE}(\log \cdot \log^{(c)}) \cap \mathrm{P})$ and a function $g \in \mathrm{FL}$ such that for all x it holds that $c_S(g(x)) \neq c_S(f(g(x)))$ and $g(x) \in \{x, f(x)\}$. We show that $A \cap S$ and $A \cap \overline{S}$ are $\leq_{\text{2-dtt}}^{\log}$-complete for $\mathcal{C}$.

Note that $(\mathrm{DSPACE}(\log \cdot \log^{(c)}) \cap \mathrm{P})$ is closed under complementation, hence we have $S \in \mathcal{C}$ and $\overline{S} \in \mathcal{C}$. Since $\mathcal{C}$ is closed under intersection, we obtain $A \cap S \in \mathcal{C}$ and $A \cap \overline{S} \in \mathcal{C}$. So it remains to show the $\leq_{\text{2-dtt}}^{\log}$-hardness of $A \cap S$ and $A \cap \overline{S}$ for $\mathcal{C}$. Since A is $\leq_{\mathrm{m}}^{\log}$-hard for $\mathcal{C}$, it suffices to show $A \leq_{\text{2-dtt}}^{\log} A \cap S$ and $A \leq_{\text{2-dtt}}^{\log} A \cap \overline{S}$. Observe that $c_A(x) = c_A(g(x)) = c_A(f(g(x)))$

and $\{g(x), f(g(x))\} \cap S \neq \emptyset$. Let $h(x) := \{g(x), f(g(x))\}$. If $x \in A$, then $h(x) \subseteq A$, hence $h(x) \cap (S \cap A) = (h(x) \cap A) \cap S = h(x) \cap S \neq \emptyset$. If $x \notin A$, then $h(x) \cap (A \cap S) \subseteq h(x) \cap A = \emptyset$. Hence, h shows that $A \leq^{\log}_{\text{2-dtt}} A \cap S$. Analogously, h shows that $A \leq^{\log}_{\text{2-dtt}} A \cap \overline{S}$. So, A is $\leq^{\log}_{\text{m}}$-complete, and $A \cap S$ and $A \cap \overline{S}$ are $\leq^{\log}_{\text{2-dtt}}$-complete for $\mathcal{C}$. This in particular implies that A is weakly $\leq^{\log}_{\text{2-dtt}}$-mitotic. □

Recall that by Theorem 10.3.1, every $\leq^{\log}_{\text{m}}$-complete set for NEXP is $\leq^{\log}_{\text{m}}$-autoreducible. Since NEXP satisfies the requirements of Theorem 11.2.1, all $\leq^{\log}_{\text{m}}$-complete sets for NEXP are weakly $\leq^{\log}_{\text{2-dtt}}$-mitotic. Moreover, Buhrman [Buh93] shows that all $\leq^{\text{p}}_{\text{1-tt}}$-complete sets for NEXP are even $\leq^{\text{p}}_{\text{m}}$-complete for NEXP. His proof also shows that all $\leq^{\log}_{\text{1-tt}}$-complete sets for NEXP are $\leq^{\log}_{\text{m}}$-complete for NEXP, so even all $\leq^{\log}_{\text{1-tt}}$-complete sets for NEXP are weakly $\leq^{\log}_{\text{2-dtt}}$-mitotic.

Corollary 11.2.2 *1. Every $\leq^{\log}_{\text{m}}$-complete set for* NEXP *is weakly $\leq^{\log}_{\text{2-dtt}}$-mitotic.*
2. Every $\leq^{\log}_{\text{1-tt}}$-complete set for NEXP *is weakly $\leq^{\log}_{\text{2-dtt}}$-mitotic.*

11.3 Truth-Table Complete Sets with One Query

We generalize the approach that we used in the last section to truth-table autoreductions that ask exactly one query. We can think of such a truth-table autoreduction for some set A as two functions f and f', where f' maps to the set of all unary Boolean functions, such that for all x it holds that $f(x) \neq x$ and $c_A(x) = f'(x)(c_A(f(x)))$. Note that we can modify the autoreduction such that f' never maps to a constant function. We construct an autoreduction that on each input either has a few successive elements in its trace on which it behaves like a many-one autoreduction, or ends up after a few steps in a small cycle. We treat cycles directly and proceed on many-one parts of the trace similar to the previous section.

Theorem 11.3.1 *Let $c \geq 1$ and let $\mathcal{C} \supseteq (\text{DSPACE}(\log \cdot \log^{(c)}) \cap \text{P})$ be closed under intersection. If A is $\leq^{\log}_{\text{1-tt}}$-complete for $\mathcal{C}$ and $\leq^{\log}_{\text{1-tt}}$-autoreducible, then A is weakly $\leq^{\log}_{\text{2-tt}}$-mitotic.*

Proof Since A is $\leq^{\log}_{\text{1-tt}}$-autoreducible, there are functions f and $f' \in \text{FL}$ such that f' maps to the set of all unary Boolean functions, and for all x it holds that $x \neq f(x)$ and $c_A(x) = f'(x)(c_A(f(x)))$. Note that we can assume that A is non-trivial (otherwise the result trivially holds), and hence that further $f'(x) \in \{\text{id}, \text{not}\}$ for all x. We define a function g by

$$g(x) := f^{(k)}(x) \quad \text{for } k \leq 3 \text{ minimal with } x \neq f^{(k)}(x) \text{ and } c_A(x) = c_A(f^{(k)}(x)), \text{ if such a } k \text{ exists, and} \tag{11.6}$$

$$g(x) := f(x) \quad \text{otherwise.} \tag{11.7}$$

Observe that $g \in \text{FL}$ (because we can decide $c_A(x) = c_A(f^{(k)}(x))$ for $k \leq 3$ by looking at f and f') and $g(x) \neq x$ for all x. Furthermore, for every x we have the following observation:

$$c_A(x) = c_A(g(x)) \iff g(x) \text{ is defined according to (11.6)} \tag{11.8}$$

For the remainder of the proof, for every x and i, we denote $x_i := g^{(i)}(x)$.

Claim 11.3.2 *For every x, at least one of the following holds:*

- $x_2 = x_4$
- $c_A(x_k) = c_A(x_{k+1}) = c_A(x_{k+2})$ *for some $k \leq 2$*

Proof Suppose for some x, the claim does not hold, hence $x_2 \neq x_4$ (and so $x_1 \neq x_3$ and $x_0 \neq x_2$), and the following holds:

$$c_A(x_k) = c_A(x_{k+1}) \implies c_A(x_{k+1}) \neq c_A(x_{k+2}), \text{ for all } k \leq 2 \tag{11.9}$$

We distinguish the following cases and show that each case leads to a contradiction.

Case 1: $c_A(x_k) \neq c_A(x_{k+1})$ and $c_A(x_{k+1}) \neq c_A(x_{k+2})$ for some $k \leq 2$. This means that $c_A(x_k) \neq c_A(g(x_k))$ and $c_A(x_{k+1}) \neq c_A(g(x_{k+1}))$. By (11.8), this means that $g(x_k)$ and $g(x_{k+1})$ are defined according to (11.7). Hence we have $g(x_k) = f(x_k)$ and $g(x_{k+1}) = f(x_{k+1})$. This means that $x_{k+2} = g(x_{k+1}) = f(x_{k+1}) = f(g(x_k)) = f(f(x_k)) = f^{(2)}(x_k)$.

We now argue that $x_{k+2} = x_k$. Suppose this is not the case, hence $x_{k+2} \neq x_k$. By our hypothesis, $c_A(x_{k+2}) = c_A(x_k)$. So $g(x_k)$ is defined according to (11.6), hence $c_A(x_k) = c_A(g(x_k))$. This contradicts $c_A(x_k) \neq c_A(g(x_k))$, so we have $x_{k+2} = x_k$.

Since $k \leq 2$, this contradicts either $x_0 \neq x_2$, $x_1 \neq x_3$, or $x_2 \neq x_4$. Hence this case cannot occur.

Case 2: $c_A(x_k) = c_A(x_{k+1})$ or $c_A(x_{k+1}) = c_A(x_{k+2})$ for all $k \leq 2$. If $c_A(x_0) \neq c_A(x_1)$, then $c_A(x_1) = c_A(x_2)$, hence $c_A(x_2) \neq c_A(x_3)$ by (11.9). If $c_A(x_0) = c_A(x_1)$, then $c_A(x_1) \neq c_A(x_2)$ by (11.9), hence $c_A(x_2) = c_A(x_3)$, and hence $c_A(x_3) \neq c_A(x_4)$ by (11.9). So in either case there exists some $k \leq 1$ such that $c_A(x_k) \neq c_A(x_{k+1})$, $c_A(x_{k+1}) = c_A(x_{k+2})$, and $c_A(x_{k+2}) \neq c_A(x_{k+3})$. So, $g(x_{k+1})$ is defined according to (11.6), and $g(x_k)$ and $g(x_{k+2})$ are defined according to (11.7). Hence we have $g(x_k) = f(x_k)$ and $g(x_{k+2}) = f(x_{k+2})$.

We first argue that $g(x_{k+1}) = f(x_{k+1})$ holds. So suppose that $g(x_{k+1}) \neq f(x_{k+1})$. We show that this implies a contradiction. Since $g(x_{k+1})$ is defined according to (11.6), but $g(x_{k+1}) \neq f(x_{k+1})$ holds, we know that $c_A(x_{k+1}) \neq c_A(f(x_{k+1}))$. Recall that $c_A(x_k) \neq c_A(x_{k+1})$, hence we obtain $c_A(x_k) = c_A(f(x_{k+1})) = c_A(f(g(x_k))) = c_A(f(f(x_k))) = c_A(f^{(2)}(x_k))$. We now argue that both $x_k \neq f^{(2)}(x_k)$ and $x_k = f^{(2)}(x_k)$ lead to a contradiction. If $x_k \neq f^{(2)}(x_k)$, then $g(x_k)$ is defined according to (11.6), which contradicts $c_A(x_k) \neq c_A(x_{k+1})$. If $x_k = f^{(2)}(x_k)$, then $x_k = x_{k+2}$, which contradicts either $x_0 \neq x_2$ or $x_1 \neq x_3$. So in both cases we obtain a contradiction, hence $g(x_{k+1}) = f(x_{k+1})$. This means that $x_{k+3} = g^{(3)}(x_k) = f^{(3)}(x_k)$.

We now argue that $x_{k+3} = x_k$. Suppose this is not the case, hence $x_{k+3} \neq x_k$. From $c_A(x_k) \neq c_A(x_{k+1}) = c_A(x_{k+2}) \neq c_A(x_{k+3})$ we obtain $c_A(x_k) = c_A(x_{k+3})$. So $g(x_k)$ is defined according to (11.6). But this contradicts $c_A(x_k) \neq c_A(x_{k+1})$, hence $x_{k+3} = x_k$.

This means that $f^{(2)}(x_{k+2}) = f(f(x_{k+2})) = f(g(x_{k+2})) = f(x_{k+3}) = f(x_k) = x_{k+1}$. Recall that $x_{k+2} = g(x_{k+1}) = f(x_{k+1})$. Since f is an autoreduction, we have $x_{k+1} \neq x_{k+2}$. Furthermore, we have $c_A(x_{k+1}) = c_A(x_{k+2})$. So $g(x_{k+2})$ is defined according to (11.6). This contradicts $c_A(x_{k+2}) \neq c_A(g(x_{k+2}))$. Hence this case cannot occur. □

From Lemma 11.1.9 we obtain a set $S \in (\mathrm{DSPACE}(\log \cdot \log^{(c)}) \cap \mathrm{P})$ and a function $h \in \mathrm{FL}$ such that

$$c_S(h(x)) \neq c_S(g(h(x))) \tag{11.10}$$

$$h(x) \in \{x, g(x)\} \tag{11.11}$$

for all x. Define the set

$$S' := \{x \in S \mid x \neq g^{(2)}(x)\} \cup \{x \mid x = g^{(2)}(x) \text{ and } x < g(x)\}$$

and observe that also $S' \in (\mathrm{DSPACE}(\log \cdot \log^{(c)}) \cap \mathrm{P})$ holds. We will show that $A \cap S'$ and $A \cap \overline{S'}$ are $\leq^{\log}_{2\text{-tt}}$-complete for $\mathcal{C}$.

Note that $(\mathrm{DSPACE}(\log\cdot\log^{(c)})\cap \mathrm{P})$ is closed under complement, thus $S', \overline{S'} \in \mathcal{C}$. Since $\mathcal{C}$ is closed under intersection, we obtain $A\cap S' \in \mathcal{C}$ and $A \cap \overline{S'} \in \mathcal{C}$. So it remains to argue for the $\leq^{\log}_{\text{2-tt}}$-hardness of $A\cap S'$ and $A\cap\overline{S'}$ for $\mathcal{C}$. Since A is $\leq^{\log}_{\text{1-tt}}$-hard for $\mathcal{C}$, it remains to show $A \leq^{\log}_{\text{2-tt}} A\cap S'$ and $A \leq^{\log}_{\text{2-tt}} A\cap\overline{S'}$. We show $A \leq^{\log}_{\text{2-tt}} A\cap S'$. Let x be some input. We distinguish the following cases:

Case 1: $x_4 = x_6$. Recall that $x_4 \neq x_5$. Let $i \in \{0,1\}$ such that $x_{4+i} < x_{5-i}$. Then we have $x_{4+i} \in S'$. We distinguish the following two subcases:

- $c_A(x) = c_A(x_{4+i})$. Then we have $c_A(x) = c_{A\cap S'}(x_{4+i})$.
- $c_A(x) \neq c_A(x_{4+i})$. Then we have $c_A(x) = 1 - c_A(x_{4+i}) = 1 - c_{A\cap S'}(x_{4+i})$.

In logspace we can determine i and find out which subcase holds by a constant number of applications of f and f'.

Case 2: $x_4 \neq x_6$. Hence for all $k \leq 4$ it holds that $x_k \neq x_{k+2}$, which in particular means $c_{S'}(x_k) = c_S(x_k)$ for all $k \leq 4$, and $x_2 \neq x_4$. By Claim 11.3.2 it holds that $c_A(x_k) = c_A(x_{k+1}) = c_A(x_{k+2})$ for some minimal $k \leq 2$. Let $y = h(x_k)$ and $z = g(y)$. From (11.11) we obtain $c_A(x_k) = c_A(y) = c_A(z)$, and from (11.10) we obtain $c_S(y) \neq c_S(z)$. Since $y, z \in \{x_0, \ldots, x_4\}$ we have $c_{S'}(y) = c_S(y)$ and $c_{S'}(z) = c_S(z)$, which means that $c_{S'}(y) \neq c_{S'}(z)$. We distinguish the following subcases.

- $c_A(x) = c_A(x_k)$. Then, $c_A(x) = c_A(y) = c_A(z)$, hence $c_A(x) = \max\{c_{A\cap S'}(y), c_{A\cap S'}(z)\}$.
- $c_A(x) \neq c_A(x_k)$. Then we have $1 - c_A(x) = c_A(y) = c_A(z) = \max\{c_{A\cap S'}(y), c_{A\cap S'}(z)\}$ and hence $c_A(x) = 1 - \max\{c_{A\cap S'}(y), c_{A\cap S'}(z)\}$.

In logspace we can determine k and find out which subcase holds by a constant number of applications of f and f'.

Hence in logspace we can determine $c_A(x)$ with at most two nonadaptive queries to $A\cap S'$. This shows $A \leq^{\log}_{\text{2-tt}} A\cap S'$. The proof for $A \leq^{\log}_{\text{2-tt}} A\cap\overline{S'}$ works analogously. So overall we obtain that A is $\leq^{\log}_{\text{1-tt}}$-complete for $\mathcal{C}$ and that $A\cap S'$ and $A\cap\overline{S'}$ are $\leq^{\log}_{\text{2-tt}}$-complete for $\mathcal{C}$. This in particular means that A is weakly $\leq^{\log}_{\text{2-tt}}$-mitotic. $\square$

Corollary 11.3.3 *Let $c \geq 1$ and let $\mathcal{C} \supseteq (\mathrm{DSPACE}(\log\cdot\log^{(c)})\cap\mathrm{P})$ be closed under intersection and complementation. If A is $\leq^{\log}_{\mathrm{m}}$-complete for $\mathcal{C}$, then A is weakly $\leq^{\log}_{\text{2-tt}}$-mitotic.*

Proof For every trivial set, the conclusion holds, so consider a non-trivial set A. If A is $\leq^{\log}_{\mathrm{m}}$-complete for $\mathcal{C}$, then A is $\leq^{\log}_{\text{1-tt}}$-complete for $\mathcal{C}$. Moreover, by Proposition 7.4.5 we know that A is $\leq^{\log}_{\text{1-tt}}$-autoreducible. From Theorem 11.3.1 we obtain that A is weakly $\leq^{\log}_{\text{2-tt}}$-mitotic. $\square$

Corollary 11.3.4 *Every $\leq^{\log}_{\mathrm{m}}$-complete set for* P *and every $\leq^{\log}_{\mathrm{m}}$-complete set for the levels Δ^{p}_k of the polynomial-time hierarchy is weakly $\leq^{\log}_{\text{2-tt}}$-mitotic.*

11.4 Disjunctive Truth-Table Complete Sets for PSPACE

We further generalize our approach to $\leq_{\mathrm{dtt}}^{\mathrm{p}}$-complete sets for PSPACE. So suppose we consider the reduction graph of some disjunctive truth-table autoreduction f for the set A. If we grant the separator enough resources, then for each input x, it can determine the smallest $y \in f(x)$ such that $c_A(x) = c_A(y)$ and hence treat f like a many-one reduction. Recall from Chapter 10 that for many complexity classes with sufficiently high computational power we obtain mitoticity of complete sets by diagonalization techniques. For PSPACE, this works as long as we consider logspace complete sets, but the simulation of arbitrary polynomial-time reductions might require more than polynomial space. Consequently, only $\leq_{\mathrm{dtt}}^{\mathrm{p}}$-autoreducibility of $\leq_{\mathrm{dtt}}^{\mathrm{p}}$-complete sets for PSPACE is known. We show weak $\leq_{\mathrm{dtt}}^{\mathrm{p}}$-mitoticity of these sets.

Lemma 11.4.1 *Let $A \in \mathrm{PSPACE}$ and let $f \in \mathrm{FP}$ be a $\leq_{\mathrm{dtt}}^{\mathrm{p}}$-autoreduction for A such that f never maps to the empty set. Then there exists a set $S \in \mathrm{PSPACE}$ such that for all x there exist $y \in f(x)$ and $z \in f(y)$ with the following properties:*

1. $c_A(x) = c_A(y) = c_A(z)$
2. $\emptyset \neq (\{x, y, z\} \cap S) \neq \{x, y, z\}$

Proof We consider the function g with

$$g(x) := \begin{cases} y_i & \text{if } f(x) = \langle y_1, \dots, y_k \rangle \wedge y_i \in A \wedge y_j \notin A \text{ for all } j < i \text{ , and} \\ y_1 & \text{if } f(x) = \langle y_1, \dots, y_k \rangle \wedge y_j \notin A \text{ for all } j \leq k, \end{cases}$$

for all x. Since $A \in \mathrm{DSPACE}(p)$ for some polynomial p, there exists a polynomial q such that $g \in \mathrm{FSPACE}(q)$. Furthermore, since g maps to values of f, we have $g(x) \neq x$ for all x, and we can modify q such that $|g(x)| \leq q(|x|)$.

We apply Lemma 11.1.9 and obtain a set $S \in \mathrm{DSPACE}(q \cdot \log^{(c)}) \subseteq \mathrm{PSPACE}$ (where $c \geq 1$ is some constant) and a function h such that $h(x) \in \{x, g(x)\}$ and $c_S(h(x)) \neq c_S(g(h(x)))$ for all x. Choose $y := g(x)$ and $z := g(y)$. Hence $y \in f(x)$ and $z \in f(y)$, and $c_A(x) = c_A(y) = c_A(z)$. Furthermore, $h(x) \in \{x, y\}$, so we either have $c_S(x) \neq c_S(y)$, or $c_S(y) \neq c_S(z)$. □

Theorem 11.4.2 *Let $k \geq 2$ and $l = k^3 + k^2 + k$.*

1. *All $\leq_{k\text{-dtt}}^{\mathrm{p}}$-complete sets for PSPACE are weakly $\leq_{l\text{-dtt}}^{\mathrm{p}}$-mitotic.*
2. *All $\leq_{\mathrm{bdtt}}^{\mathrm{p}}$-complete sets for PSPACE are weakly $\leq_{\mathrm{bdtt}}^{\mathrm{p}}$-mitotic.*
3. *All $\leq_{\mathrm{dtt}}^{\mathrm{p}}$-complete sets for PSPACE are weakly $\leq_{\mathrm{dtt}}^{\mathrm{p}}$-mitotic.*

Proof If L is $\leq_{k\text{-dtt}}^{\mathrm{p}}$-complete for PSPACE, then L is $\leq_{k\text{-dtt}}^{\mathrm{p}}$-autoreducible [GOP+07]. Let f be some $\leq_{k\text{-dtt}}^{\mathrm{p}}$-autoreduction for L. We assume that f never maps to the empty set. From Lemma 11.4.1 we obtain $S \in \mathrm{PSPACE}$ with the specified properties. We show that $L \cap S$ is $\leq_{l\text{-dtt}}^{\mathrm{p}}$-complete for PSPACE, the completeness of $L \cap \overline{S}$ is shown analogously.

Recall that PSPACE is closed under intersection, so $L \cap S \in \mathrm{PSPACE}$, and it remains to show hardness. Let $m = k^2 + k + 1$, hence $l = k \cdot m$. For arbitrary $A \in \mathrm{PSPACE}$ we already know that $A \leq_{k\text{-dtt}}^{\mathrm{p}} L$, hence it suffices to show $L \leq_{m\text{-dtt}}^{\mathrm{p}} L \cap S$. On input x, return $Q_x := \{x\} \cup f(x) \cup \bigcup_{y \in f(x)} f(y)$, which can be computed in polynomial time. The number of the elements in the output is bounded by $1 + k + k^2 = m$. To show that Q_x is a reduction as claimed above, choose y, z as in the lemma. If $x \in L$, then $\{x, y, z\} \subseteq L$. So from $\{x, y, z\} \cap S \neq \emptyset$ and $\{x, y, z\} \subseteq Q_x$ we obtain $(L \cap S) \cap Q_x \supseteq (L \cap S) \cap \{x, y, z\} = S \cap \{x, y, z\} \neq \emptyset$. If $x \notin L$, then $(L \cap S) \cap Q_x \subseteq L \cap Q_x = \emptyset$. This shows the $\leq_{l\text{-dtt}}^{\mathrm{p}}$-hardness.

We have shown item 1. The other items are shown analogously. □

11.5 Summary and Discussion

We showed that for many complexity classes, $\leq_{\mathrm{m}}^{\log}$-autoreducibility of $\leq_{\mathrm{m}}^{\log}$-complete sets implies weak $\leq_{2\text{-dtt}}^{\log}$-mitoticity, and an analogous result holds for $\leq_{1\text{-tt}}^{\log}$-reducibility. In the proof, we considered the autoreduction graph of an autoreduction function f. We adapted the deterministic coin tossing technique by Cole and Vishkin [CV86] and showed that it suffices to follow the trace of f for very few steps to obtain a splitting of A into two equivalent parts. In order to obtain logspace equivalence, we had to shift some complexity into the separator set, and since the separator became more complex, we only obtained weak mitoticity.

Recall from the last chapters that $\leq_{\mathrm{m}}^{\log}$-complete sets for P and Δ_k^{p} are $\leq_{1\text{-tt}}^{\log}$-autoreducible, and in the case of NEXP we even obtain $\leq_{\mathrm{m}}^{\log}$-autoreducibility. The question whether these results can be improved to logspace mitoticity is open. Our general results provide some progress on this question, because they show that at least weak $\leq_{2\text{-tt}}^{\log}$-mitoticity or even weak $\leq_{2\text{-dtt}}^{\log}$-mitoticity holds. It remains open whether this can be improved to $\leq_{1\text{-tt}}^{\log}$-mitoticity or even $\leq_{\mathrm{m}}^{\log}$-mitoticity.

Finally, recall from the last chapter that all $\leq_{\mathrm{dtt}}^{\log}$-complete sets for PSPACE are $\leq_{\mathrm{dtt}}^{\log}$-mitotic, which we showed by diagonalization. Note that if we consider $\leq_{\mathrm{dtt}}^{\mathrm{p}}$-complete sets for PSPACE, then this technique might not work, since we do not know whether we can simulate arbitrary polynomial-time reductions in polynomial space. However, it is known that all $\leq_{\mathrm{dtt}}^{\mathrm{p}}$-complete sets for PSPACE are $\leq_{\mathrm{dtt}}^{\mathrm{p}}$-autoreducible [GOP+07]. Following the same approach as described above we showed that all $\leq_{\mathrm{dtt}}^{\mathrm{p}}$-complete sets for PSPACE are weakly $\leq_{\mathrm{dtt}}^{\mathrm{p}}$-mitotic, and a similar result holds for all $\leq_{k\text{-dtt}}^{\mathrm{p}}$-complete sets for PSPACE. Improving this to $\leq_{\mathrm{dtt}}^{\mathrm{p}}$-mitoticity remains a challenging task for future work.

Bibliography

[AB09] S. Arora and B. Barak. *Computational Complexity - A Modern Approach.* Cambridge University Press, 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.

[ACG$^+$99] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and approximation: Combinatorial optimization problems and their approximability properties.* Springer Verlag, 1999.

[AKS12] H. C. An, R. Kleinberg, and D. B. Shmoys. Improving Christofides' algorithm for the s-t path TSP. In *STOC*, pages 875–886, 2012.

[Aro98] S. Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *J. ACM*, 45(5):753–782, 1998.

[AS84] K. Ambos-Spies. P-mitotic sets. In E. Börger, G. Hasenjäger, and D. Roding, editors, *Logic and Machines*, volume 171 of *Lecture Notes in Computer Science*, pages 1–23. Springer Verlag, 1984.

[Bal90] J. L. Balcázar. Self-reducibility. *Journal of Computer and System Sciences*, 41(3):367–388, 1990.

[Ber77] L. Berman. *Polynomial Reducibilities and Complete Sets.* PhD thesis, Cornell University, Ithaca, NY, 1977.

[BF81] J. Beck and T. Fiala. "Integer-Making" Theorems. *Discrete Applied Mathematics*, 3(1):1–8, 1981.

[BF92] R. Beigel and J. Feigenbaum. On being incoherent without being very hard. *Computational Complexity*, 2:1–17, 1992.

[BFvMT00] H. Buhrman, L. Fortnow, D. van Melkebeek, and L. Torenvliet. Separating complexity classes using autoreducibility. *SIAM Journal on Computing*, 29(5):1497–1520, 2000.

[BH77] L. Berman and J. Hartmanis. On isomorphisms and density of NP and other complete sets. *SIAM Journal on Computing*, 6(2):305–322, 1977.

[BHT98] H. Buhrman, A. Hoene, and L. Torenvliet. Splittings, robustness, and structure of complete sets. *SIAM Journal on Computing*, 27(3):637–653, 1998.

[BMP08] M. Bläser, B. Manthey, and O. Putz. Approximating multi-criteria max-TSP. In *ESA*, pages 185–197, 2008.

[BP87] F. Barahona and W. R. Pulleyblank. Exact arborescences, matchings and cycles. *Discrete Appl. Math.*, 16(2):91–99, 1987.

[BT94] H. Buhrman and L. Torenvliet. On the structure of complete sets. In *Proceedings 9th Structure in Complexity Theory*, pages 118–133, 1994.

[Buh93] H. Buhrman. *Resource Bounded Reductions.* PhD thesis, University of Amsterdam, 1993.

[Car11] C. Carathéodory. Über den Variabilitätsbereich der Fourier'schen Konstanten von positiven harmonischen Funktionen. *Rendiconti del Circolo Matematico di Palermo*, 32(1):193–217, 1911.

[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.

[CKS81] A. K. Chandra, D. Kozen, and L. J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.

[CV86] R. Cole and U. Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70(1):32–53, 1986.

[CVZ11] C. Chekuri, J. Vondrák, and R. Zenklusen. Multi-budgeted matchings and matroid intersection via dependent rounding. In D. Randall, editor, *SODA*, pages 1080–1097. SIAM, 2011.

[DS03] B. Doerr and A. Srivastav. Multicolour discrepancies. *Combinatorics, Probability & Computing*, 12(4):365–399, 2003.

[Edm65] J. Edmonds. Maximum matching and a polyhedron with 0, 1-vertices. *Journal of Research of the National Bureau of Standards B*, 69:125–130, 1965.

[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] Matthias Ehrgott. *Multicriteria Optimization (2. ed.).* Springer, 2005.

[FGL$^+$11] K. Fleszar, C. Glaßer, F. Lipp, C. Reitwießner, and M. Witek. The complexity of solving multiobjective optimization problems and its relation to multivalued functions. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:53, 2011.

[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 Symposium on Theoretical Informatics (LATIN)*, volume 7256 of *Lecture Notes in Computer Science*, pages 338–349. Springer, 2012.

[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.

[GH92] K. Ganesan and S. Homer. Complete problems and strong polynomial reducibilities. *SIAM Journal on Computing*, 21(4):733–742, 1992.

[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.

[Gla10] C. Glaßer. Space-efficient informational redundancy. *Journal of Computer and System Sciences*, 76(8):792–811, 2010.

[GLS81] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

[GLS88] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics.* Springer, 1988.

[GNR$^+$13a] C. Glaßer, D. T. Nguyen, C. Reitwießner, A. L. Selman, and M. Witek. Autoreducibility of complete sets for log-space and polynomial-time reductions. In *Proceedings 40th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 7965 of *Lecture Notes in Computer Science*, pages 473–484. Springer, 2013.

[GNR$^+$13b] C. Glaßer, D. T. Nguyen, C. Reitwießner, A. L. Selman, and M. Witek. Autoreducibility of complete sets for log-space and polynomial-time reductions. *Electronic Colloquium on Computational Complexity (ECCC)*, 20:47, 2013.

[GOP$^+$07] C. Glaßer, M. Ogihara, A. Pavan, A. L. Selman, and L. Zhang. Autoreducibility, mitoticity, and immunity. *Journal of Computer and System Sciences*, 73(5):735–754, 2007.

[GPSZ08] C. Glaßer, A. Pavan, A. L. Selman, and L. Zhang. Splitting NP-complete sets. *SIAM Journal on Computing*, 37(5):1517–1535, 2008.

[GRSW10a] C. Glaßer, C. Reitwießner, H. Schmitz, and M. Witek. Approximability and hardness in multi-objective optimization. In *Proceedings 6th Conference on Computability in Europe (CiE)*, volume 6158 of *Lecture Notes in Computer Science*, pages 180–189. Springer, 2010.

[GRSW10b] C. Glaßer, C. Reitwießner, H. Schmitz, and M. Witek. Hardness and approximability in multi-objective optimization. *Electronic Colloquium on Computational Complexity (ECCC)*, 17:31, 2010.

[GRW09] C. Glaßer, C. Reitwießner, and M. Witek. Improved and derandomized approximations for two-criteria metric traveling salesman. *Electronic Colloquium on Computational Complexity (ECCC)*, 16:76, 2009.

[GRW11a] 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 fuer Informatik, 2011.

[GRW11b] C. Glaßer, C. Reitwießner, and M. Witek. Applications of discrepancy theory in multiobjective approximation. *CoRR*, abs/1107.0634, 2011.

[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.

[GW13] C. Glaßer and M. Witek. Autoreducibility and mitoticity of logspace-complete sets for NP and other classes. *Electronic Colloquium on Computational Complexity (ECCC)*, 20:188, 2013.

[GW14] C. Glaßer and M. Witek. Autoreducibility and mitoticity of logspace-complete sets for NP and other classes. In *Proceedings 39th International Symposium of Mathematical Foundations of Computer Science (MFCS)*, volume 8635 of *Lecture Notes in Computer Science*, pages 311–323. Springer, 2014.

[HKR93] S. Homer, S. A. Kurtz, and J. S. Royer. On 1-truth-table-hard languages. *Theoretical Computer Science*, 115(2):383–389, 1993.

[Hoo91] J. A. Hoogeveen. Analysis of Christofides' heuristic: Some paths are more difficult than cycles. *Operations Research Letters*, 10:291–295, 1991.

[Imm88] N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17(5):935–938, 1988.

[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.

[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.

[Lad73] R. E. Ladner. Mitotic recursively enumerable sets. *Journal of Symbolic Logic*, 38(2):199–211, 1973.

[Lad75] R. E. Ladner. The circuit value problem is log space complete for P. *SIGACT News*, 7(1):18–20, January 1975.

[LL76] R. E. Ladner and N. A. Lynch. Relativization of questions about log space computability. *Mathematical Systems Theory*, 10:19–32, 1976.

[LLKS85] E. L. Lawler, J. K. Lenstra, A. H. G. Kan, and D. B. Shmoys. *The Traveling Salesman Problem.* Wiley Interscience Series in Discrete Mathematics. John Wiley & Sons, 1985.

[Lyn78] N. A. Lynch. Log space machines with multiple oracle tapes. *Theoretical Computer Science*, 6:25–39, 1978.

[Mah82] S. R. Mahaney. Sparse complete sets of NP: Solution of a conjecture of berman and hartmanis. *J. Comput. Syst. Sci.*, 25(2):130–143, 1982.

[Man05] B. Manthey. *Approximability of cycle covers and smoothed analysis of binary search trees.* PhD thesis, Universität zu Lübeck, 2005.

[Man12a] B. Manthey. Deterministic algorithms for multi-criteria Max-TSP. *Discrete Applied Mathematics*, 160(15):2277 – 2285, 2012.

[Man12b] B. Manthey. On approximating multicriteria TSP. *ACM Transactions on Algorithms*, 8(2):17, 2012.

[MR09] B. Manthey and L. S. Ram. Approximation algorithms for multi-criteria traveling salesman problems. *Algorithmica*, 53(1):69–88, 2009.

[MVV87] K. Mulmuley, U. V. Vazirani, and V. V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.

[NS12] D. T. Nguyen and A. L. Selman. Autoreducibility for NEXP. Seventh International Conference on Computability, Complexity, and Randomness, 2012.

[NS14] D. T. Nguyen and A. L. Selman. Non-autoreducible sets for NEXP. In *Proceedings 31st Symposium on Theoretical Aspects of Computer Science (STACS)*, Leibniz International Proceedings in Informatics (LIPIcs), pages 590–601. Springer Verlag, 2014.

[OW91] M. Ogiwara and O. Watanabe. On polynomial-time bounded truth-table reducibility of NP sets to sparse sets. *SIAM Journal on Computing*, 20(3):471–483, 1991.

[Pal14] K. Paluch. Better approximation algorithms for maximum asymmetric traveling salesman and shortest superstring. *CoRR*, abs/1401.3670, 2014.

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

[PEvZ12] K. Paluch, K. Elbassioni, and A. v. Zuylen. Simpler Approximation of the Maximum Asymmetric Traveling Salesman Problem. In C. Dürr and T. Wilke, editors, *29th International Symposium on Theoretical Aspects of Computer Science (STACS 2012)*, volume 14 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 501–506, Dagstuhl, Germany, 2012. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[PL86] D. Plummer and L. Lovász. *Matching Theory.* North-Holland Mathematics Studies. Elsevier Science, 1986.

[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.

[PR82] M. W. Padberg and M. R. Rao. Odd minimum cut-sets and b-matchings. *Mathematics of Operations Research*, 7(1):67–80, 1982.

[PV06] C. Papadimitriou and S. Vempala. On the approximability of the traveling salesman problem. *Combinatorica*, 26(1):101–120, 2006.

[PY82] C. H. Papadimitriou and M. Yannakakis. The complexity of restricted spanning tree problems. *J. ACM*, 29(2):285–309, 1982.

[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.

[Sav70] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177 – 192, 1970.

[Sch83] A. Schrijver. Short proofs on the matching polyhedron. *Journal of Combinatorial Theory, Series B*, 34(1):104 – 108, 1983.

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

[ST42] A. H. Stone and J. W. Tukey. Generalized "sandwich" theorems. *Duke Math J.*, 9(2):356–359, 1942.

[Sto76] L. J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.

[SW85] W. Stromquist and D. R. Woodall. Sets on which several measures agree. *Journal of Mathematical Analysis and Applications*, 108(1):241–248, 1985.

[Sze88] R. Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Inf.*, 26(3):279–284, 1988.

[Tra70] B. Trakhtenbrot. On autoreducibility. *Dokl. Akad. Nauk SSSR*, 192(6):1224–1227, 1970. Translation in Soviet Math. Dokl. 11(3): 814–817, 1970.

[Tut54] W. T. Tutte. A short proof of the factor theorem for finite graphs. *Canad. J. Math.*, 6:347–352, 1954.

[Vaz01] V. V. Vazirani. *Approximation algorithms.* Springer, 2001.

[Wag87] K. W. Wagner. More complicated questions about maxima and minima, and some closures of NP. *Theoretical Computer Science*, 51(1–2):53 – 80, 1987.

[Wra76] C. Wrathall. Complete sets and the polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):23 – 33, 1976.

Index