Output-sensitive Complexity of Multiobjective Combinatorial Optimization with an Application to the Multiobjective Shortest Path Problem

Dissertation

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

der Technischen Universität Dortmund

an der Fakultät für Informatik

von

Friedrich Konstantin Bökler

Dortmund

2018

Tag der mündlichen Prüfung:	**24. April 2018**
Dekan:	**Prof. Dr. Gernot A. Fink**
GutachterInnen:	**Prof. Dr. Petra Mutzel** (TU Dortmund, Fakultät für Informatik) **Prof. Dr. Matthias Ehrgott** (Lancaster University, Department of Management Science)

Abstract

In this thesis, we are concerned with multiobjective combinatorial optimization (MOCO) problems. We attempt to fill the gap between theory and practice, which comes from the lack of a deep complexity theory for MOCO problems. In a first part, we consider a complexity notion for multiobjective optimization problems which derives from output-sensitive complexity. The primary goal of such a complexity notion is to classify problems into easy and hard problems. We show that the multiobjective minimum cut problem as well as the multiobjective linear programming problem are easy problems in this complexity theory. We also show that finding extreme points of nondominated sets is an easy problem under certain assumptions. On the side of hard problems, there are obvious ones like the multiobjective traveling salesperson problem. Moreover, we show that the well-known multiobjective shortest path problem is a hard problem. This is also the case for the multiobjective matching and integer min-cost flow problem. We also identify a class of problems for which the biobjective unconstraint combinatorial optimization problem is a barrier for efficient solvability.

In a second part, we are again concerned with the gap between theory and practice. This time, we approach the multiobjective shortest path (MOSP) problem from the practical side. For the application in the planning of power transmission lines, we need to have implementations which can cope with large graphs and a larger number of objectives. The results from the first part suggest that exact methods might be incapable of achieving this goal which we also prove empirically. This is why we decide to study the literature on approximation algorithms for the MOSP problem. We conclude that they do not scale well with the number of objectives in general and that there are no practical implementations available. Hence, we develop a novel approximation algorithm in the second part which leans to the exact approaches which are well tested in practice. In an extensive computational study, we show that our implementation of this algorithm performs well even on a larger number of objectives. We compare our implementation to implementations of the other existing approximation algorithms and conclude that our implementation is superior on instances with more than three objectives.

Zusammenfassung

Diese Dissertation ist mit der Komplexität von mehrkriteriellen kombinatorischen Optimierungsproblemen (MOCO) befasst. Hierbei wollen wir die Lücke schließen, die sich aus dem Fehlen einer umfassenden Komplexitätstheorie dieser Probleme ergibt. In einem ersten Teil beschreiben wir einen Komplexitätsbegriff für mehrkriterielle Optimierungsprobleme, der sich von ausgabesensitiver Komplexität ableitet. Das Ziel eines Komplexitätsbegriffes ist die Klassifikation von Problemen in einfache und schwierige Probleme. Wir zeigen, dass die Probleme einen Pareto-optimalen Schnitt in einem Graphen zu bestimmen, sowie mehrkriterielle lineare Optimierung einfache Probleme nach dieser Komplexitätstheorie sind. Auch das Problem Extrempunkte von Pareto-Fronten zu bestimmen ist unter bestimmten Bedingungen ein einfaches Problem. Auf der Seite der schwierigen Probleme können wir über die offensichtlichen Probleme wie das mehrkriterielle Handlungsreisendenproblem hinaus auch zeigen, dass das bekannte mehrkriterielle Kürzeste-Wege-Problem ein schwieriges Problem darstellt. Dies gilt ebenso auch für das mehrkriterielle Zuordnungsproblem in allgemeinen Graphen und das mehrkriterielle Flussproblem mit ganzzahligen Flussvariablen. Wir finden in dieser Arbeit außerdem eine Klasse von Problemen, deren effiziente Lösbarkeit von der effizienten Lösbarkeit des bikriteriellen unrestringierten kombinatorischen Optimierungsproblems abhängt.

In einem zweiten Teil beschäftigen wir uns wieder mit der Lücke zwischen Theorie und Praxis. Diesmal nähern wir uns dem mehrkriteriellen Kürzeste-Wege-Problem von der praktischen Seite. Für eine Anwendung in der Stromtrassenoptimierung ist es nötig einen Algorithmus zu finden, der sowohl mit großen Graphen, als auch mit mehreren Zielfunktionen umgehen kann. Aus dem ersten Teil können wir ableiten, dass exakte Methoden dort an ihre Grenzen stoßen, was wir auch empirisch belegen. Wir studieren daher Approximationsalgorithmen aus der Literatur und stellen fest, dass sie in der Anzahl der Zielfunktionen nur schlecht skalieren und auch noch nicht praxiserprobt sind. Daher entwickeln wir im zweiten Teil einen neuen Approximationsalgorithmus, der sich stark an die Errungenschaften der praktischen Algorithmen orientiert. Wir zeigen in einem groß angelegten Experiment, dass unsere Implementierung des Algorithmus auch noch auf einer größeren Anzahl von Zielfunktionen praxistauglich ist. Der Vergleich mit unseren Implementierungen der existierenden Approximationsalgorithmen zeigt zudem, dass unsere Implementierung den anderen auf Instanzen mit mehr als drei Zielfunktionen überlegen ist.

Acknowledgements

First of all I want to thank the Lehrstuhl 11 at the TU Dortmund. Especially my advisor, Prof. Dr. Petra Mutzel, for always believing in my ideas, giving me the best support and opportunities to pursue my research goals; for her guidance and for running interference for all of us, leaving time for research and our personal developments as researchers. I thank Prof. Dr. Johannes Fischer for mentoring me and showing me new tracks. I thank my proof readers Dr. Nils Kriege, Denis Kurz, and Dr. Bernd Zey for giving me most valuable hints and corrections. Denis, thank you for your great company, criticism, every single counterexample, countless hours of research, presentations of ideas, and exceptionally bad jokes. Thanks to all people with whom I shared one or another chat, lunch or coffee break in all these years: Tobias Brinkjost, Andre Droschinsky, Marius Greiff, Dr. Carsten Gutwenger, Adalat Jabrayilov, Roman Kalkreuth, Dr. Dominik Kopczynski, Dominik Köppl, Dr. Nils Kriege, Prof. Dr. Johannes Fischer, Florian Kurpicz, Denis Kurz, Dr. Marcel Martin, Christopher Morris, Prof. Dr. Petra Mutzel, Prof. Dr. Boris Naujoks, Dr. Arno Pasternak, Dr. Mike Preuß, Jan Quadflieg, Prof. Dr. Sven Rahmann, Prof. Dr. Günter Rudolph, Till Schäfer, Prof. a.D. Dr.-Ing. Dr. Sc. h.c. Hans-Paul Schwefel, Christiane Spisla, Henning Timm, Dr. Igor Vatolkin, Vanessa Volz, Dr. Simon Wessing, and Dr. Bernd Zey. And last, but certainly not least, Gundel Jankord: Without you, the whole thing would capsize for sure!

In my years visiting research conferences in the field of multiobjective optimization, I had the opportunity to meet many wonderful people to share ideas with as well as good times. You were so kind to adopt me instantly and I am very happy that many friendships evolved through this. I want to thank Prof. Dr. Kathrin Klamroth and her whole group for your hospitality, kindness, and many ideas I got through very thoughtful discussions. And also MCF Dr Anthony Przybylski for your original ideas, support, and discussions—not exclusively research related. I want to thank Prof. Dr. Matthias Ehrgott for believing in my ideas and for agreeing to co-examine this dissertation. And also Prof. Dr. Andreas Löhne and his group for many fruitful discussions on optimization, polytopes and the Knigge. I thank Prof. Dr. Paolo Serafini for thoughtful discussions on **NP**-hardness in multiobjective optimization.

I thank the researchers from my practical project on power transmission line planning, Daniel Bachmann, Dr. Mike Dokter, Dr. Anna Ilyina, Jakob Kopec, Dr. Björn Schwarze, and Dr. Frank Weichert, for collaborating on a very important problem that needs more attention from the scientific community.

I want to thank Prof. Dr. Christoph Buchheim, Prof. Dr. Markus Chimani, Dr. Jannis Kurtz, Dr. Sven Mallach, and Dr. Daniel Schmidt for having open ears, eagerness in thinking through ideas and your support.

I also want to express my deep gratitude to my parents, Trixi and Sigi, who made everything possible in all those years. Especially, that I never had to worry about money or other administrative things and this was not taken for granted.

And thank you, Svenja, for bringing me right back on track several times and for many more things I feel incapable of expressing here.

Contents

1. Introduction **1**

2. Definitions and Notation **7**
2.1. Sets, Functions, Relations and Orders 7
2.2. Linear Algebra, Topology, and Polyhedra 8
2.3. Graphs 9
2.4. Computational Complexity and Encoding 10
2.5. Multiobjective Optimization 15

I. Output-Sensitive Complexity of Multiobjective Optimization **21**

3. Introduction **23**
3.1. Previous Work 23
3.2. Contributions and Organization 25

4. Preliminaries **27**
4.1. Traditional **NP**-Hardness Theory 27
4.2. Output-Sensitive Complexity Theory 32
4.3. Connection to Smoothed Analysis for MOCO Problems 35
4.4. A Positive Example 36

5. Multiobjective Linear Optimization and Nondominated Extreme Points **39**
5.1. The Benson Algorithm Family 42
5.2. Finding Facets: Benson's Algorithm 44
5.3. Finding Extreme Points: The Dual of Benson's Algorithm 55
5.4. Application to Multiobjective Combinatorial Optimization 61

6. Showing Hardness **69**
6.1. Multiobjective Shortest Path 69
6.2. BUCO and **BUCO**-Hardness 73

7. Conclusion **81**

II. The Multiobjective Shortest Path Problem in Practice **83**

8. Introduction **85**
8.1. Contributions and Organisation 86

9. Preliminaries **87**
9.1. Multiobjective Labeling Algorithms 87
9.2. Instances 90
9.3. Node Selection vs. Label Selection 91
9.4. Tree-Deletion Pruning 95

10. Multiobjective Approximation **103**
10.1. Models of Approximation 103
10.2. Literature Review 104
10.3. A General Approximation Scheme 107
10.4. Comparison of Practical Running Time and Ratio 111

11. Conclusion **125**

Bibliography **127**

III. Appendix **137**

Index **139**

List of Figures **143**

List of Tables **145**

List of Algorithms **147**

1. Introduction

The output is exponential, end of discussion

—Jack Edmonds

If we follow the history of the theory of algorithms, we can observe several different approaches to the question when an algorithm is called efficient. In the beginning of the investigation of the efficiency of algorithms, an algorithm was regarded as being efficient if its running time is finite. This rendered all combinatorial optimization problems easy problems as the number of solutions of an instance is finite and a simple search of all solutions always gives us an optimal solution.

But this notion was of no use for practitioners: If we really are interested in finding an optimal solution for the given traveling salesperson problem instance at hand, knowing that we can find an optimal solution by searching all feasible tours is of no help for us. We do not only want to use a finite algorithm, we want to use a fast algorithm for our problem. Following the seminal works by A. Cobham [33] and J. Edmonds [53], a computational problem is regarded as efficiently solvable or tractable if it can be solved in polynomial time today. This thesis is known as the Cobham-Edmonds thesis.

This view made it potentially possible to classify problems by their complexity as being easy or hard. While proofs of the polynomial complexity and thus "easiness" of computational problems were already known at that time, a proof for a computable problem not being polynomial-time solvable was not in sight.

This was especially disappointing to the community trying to solve traveling salesperson (TSP) problem instances. The TSP problem seemed to be notoriously hard but still could not be proven to be a hard problem in the sense of the Cobham-Edmonds thesis. The work by Cook [35], Karp [82], and Levin [94] opened ways to circumvent this issue. The observation that many decision problems related to interesting combinatorial optimization problems were easy to verify if a certificate was given culminated in the definition of the complexity class **NP**. Moreover, the observation that many of these decision problems were related in a polynomial way led to the definition of **NP**-hardness and **NP**-completeness. Consequently, many problems and also the decision version of the TSP were classified as **NP**-complete. This notion did not solve the problem of not being able to show that the TSP is a hard problem in the sense of the Cobham-Edmonds thesis. Nevertheless, more and more problems were proven to be **NP**-complete and thus, a polynomial time algorithm for these problems got even more unlikely. The question if there are polynomial time algorithms for **NP**-complete problems is the now very famous question if $\mathbf{P} = \mathbf{NP}$ and is one of the Millennium

Problems which were stated by the Clay Institute in 2000[1]. So, the question whether a problem is easy or hard boiled down to the question if it is polynomial-time solvable or **NP**-hard. The hope was to guide practitioners in selecting efficient algorithms for their problem at hand.

But in some cases, the theory could not satisfy the practitioners. One example is the simplex algorithm [39] for linear programming (LP) which does not have a polynomial running time in general and thus does not classify LP as a polynomial-time solvable problem. In the language of the Cobham-Edmonds thesis, it is not a good or efficient algorithm. After the works by Karmarkar [81] and Khachiyan [84], we do know polynomial-time algorithms for LP. Nevertheless, the algorithms which are mostly used in practical implementations for solving LP are based on the simplex algorithm. Arguably, the Cobham-Edmonds thesis fails when we want to select an algorithm for LP.

One attempt to conquer the realm of practical algorithms is *algorithm engineering* (cf. also [32]). In this methodology which leans more to the engineering side of algorithms, implementations of algorithms are considered first class citizens of our research. With a specific problem at hand, we can tailor custom made algorithms (and subsequent implementations) to exactly the real-world problem we have. Analysis of algorithms still is a major concern but so is testing our implementations and benchmarking them on thoughtfully chosen instance sets and case studies. Algorithm engineering can be seen as a step back from purely theory driven design to a broader view where theoretical running times are as important as the performance on real-world data. And also the empirical results give feedback to the theory: In an attempt to close the gap between the classical Cobham-Edmonds thesis and the observations made in computational experiments, several new running time frameworks have been developed. Examples for this are smoothed analysis (cf., e.g., Spielman and Teng [129]) and external memory algorithms (cf., e.g., Aggarwal and Vitter [2]).

A new situation emerged from multiobjective optimization (MOO). The most basic problems as the multiobjective shortest path, spanning tree, or assignment problem call for output sizes which can be super-polynomial in the input size. Regarding complexity theory in the sense of the Cobham-Edmonds thesis, in 1980, C. H. Papadimitriou remembers J. Edmonds saying: "The output is exponential, end of discussion" [109]. And in that spirit, the research for efficient exact algorithms for MOO problems was continued till today. Running time analyses are usually not conducted with the reasoning that the problems are intractable. Instead, approximation algorithms are developed which are not tested in practice and the best algorithms in practice are inefficient ones in the sense of the Cobham-Edmonds thesis. Consequently, we have a similar picture to the time before the Cobham-Edmonds thesis: Instead of all problems being easy, now all problems are hard which is again a problem for practitioners.

We illustrate the consequences of this situation on an example: the multiobjective shortest path problem (MOSP). Beginning in the end of the 1970s, researchers from the field of operations research investigated algorithms and implementations for the

[1] http://www.claymath.org/millennium-problems, last accessed: Jan. 11th 2018.

MOSP problem. The most successful implementations come from a labeling idea similar to Dijkstra's algorithm: We store a label for every path we already investigated at the node this paths ends in. The label consists of this path's cost. We can extend these paths by creating new labels at the heads of all edges going out of the node, setting the cost stored in the new label to the cost of the old label plus the cost of the edge. A basic question is: Given a concrete labeling algorithm, how many labels do we need to find a representative set of paths at the target node?

The classical answer is the following: Since the size of the nondominated set or Pareto-front can be exponential in the input size, we need an exponential number of labels in the worst-case. But what happens for instances with a small sized nondominated set? Not every instance is a badly behaving worst-case instance, and in fact, results from smoothed analysis [26] and from empirical studies suggest that this is rarely the case. Are there examples on which we have small nondominated sets, but still need a large number of labels? Or to put it in a more general perspective: Can we have fast algorithms if the nondominated set is small and not to slow if the nondominated set is large?

In the first part of this thesis, we give answers to this question and to other questions concerned with the computational complexity of MOO problems, especially multiobjective combinatorial optimization (MOCO) problems. Concerning the MOSP problem, we see that in a labeling algorithm, if the nondominated set contains at least two points then it can happen that we still need a super-polynomial number of labels, independent of the size of the nondominated set and no matter how smart we are in improving our algorithm (unless $\mathbf{P} = \mathbf{NP}$). This is possible by introducing a new running-time analysis framework derived from output-sensitive complexity theory. We give examples for easy problems, e.g., the multiobjective global minimum cut problem or the multiobjective linear programming problem, and of hard problems, e.g., the multiobjective shortest path problem and the multiobjective matching problem.

We are also concerned with finding extreme points of the nondominated set of MOCO problems. In the biobjective case, it is often easy to find a certain subset of the nondominated set, called the extreme points of the nondominated set. Though, a general methodology for more than two objectives is missing. We show that using our analysis framework and by showing that multiobjective linear programming is easy under certain assumptions, we can find the extreme points for every MOCO problem efficiently, as long as the MOCO problem fulfills certain requirements. This answers an open problem by Ehrgott and Gandibleux [57] from 2000, addressing this as a "first step to an application of the two phases method in three or more criteria MOCO".

The second part of this thesis is concerned with solving the multiobjective shortest path problem in practice. We introduce a problem from practice for which it is crucial to be able to solve shortest path instances with more than four or five objectives. As we learned from the first part, if the instances become larger we cannot expect our labeling algorithms to work well so we need other methods. The literature on algorithms for the multiobjective shortest path problem does provide approximation algorithms for harder cases, but they are not tested in practice. In fact, the best-case running times depend exponentially on the number of objectives. In the second part,

we introduce an approximation algorithm to remedy this condition. Although the worst-case running time is worse than the fastest known approximation algorithm, we show in extensive empirical tests that it is the fastest approximation algorithm available for practical purposes.

Organization

This thesis is divided into two parts: Part I introduces the complexity framework of output-sensitive complexity to multiobjective optimization and we see examples of easy and hard problems. In Part II, we present a novel approximation algorithm for the MOSP problem and show that its running time is much less dependent on the number of objectives than the known algorithms in the literature.

Before the first part, in Chapter 2 we introduce the necessary definitions and notation necessary to read the rest of this thesis. Most of the definitions are standard, but in some cases as the max and min operators, we change semantics a slight bit. We also introduce the problems we investigate in this thesis for reference.

In Chapter 3, the first chapter of Part I, we introduce the complexity notions known for multiobjective optimization. We show the results and limits of these theories and also give an overview of the general findings regarding output-sensitive complexity in this thesis.

A critical investigation of **NP**-hardness in multiobjective optimization is the first part of Chapter 4. This is the main contribution of a paper that appeared at the International Conference on Evolutionary Multi-Criterion Optimization in 2017 (cf. Bökler [17]). We also give a formal definition of output-sensitive complexity there, discuss the connection to smoothed analysis and give the first positive example of an easy MOCO problem in the framework of output-sensitive complexity.

Chapter 5 is dedicated to multiobjective linear programming (MOLP). We prove that under mild assumptions, MOLP is also an easy problem and investigate the connection to multiobjective combinatorial optimization. An extended abstract of the part regarding the Dual Benson algorithm and multiobjective combinatorial optimization appeared at the European Symposium on Algorithms in 2015 (cf. Bökler and Mutzel [19]).

Following the positive results in the last two sections, Chapter 6 is concerned with hardness in the framework of output-sensitive complexity. More specifically, we prove that the MOSP problem is a hard enumeration problem which is a main contribution of the author in a paper that appeared in the Journal of Multi-Criteria Decision Analysis in 2017 (Bökler et al. [18]). For some problems, we cannot determine the complexity status as yet but we introduce a reduction among MOO problems and show relationships between several well-known MOCO problems.

Chapter 7 concludes the first part with a summary of the main points and open problems.

The second part commences with Chapter 8 in which we present a problem concerned with power transmission line optimization. The problem is an example of a MOSP problem in which more than just a few objectives are present and the instances are

fairly large.

In Chapter 9, we are concerned with preliminary observations regarding the MOSP problem. We present results regarding common believes in computational studies on the MOSP problem and introduce a new pruning heuristic for label-correcting algorithms. These results are published at the Workshop on Algorithms and Computation in 2017 (cf. Bökler and Mutzel [20]).

The approximation algorithm is then introduced in Chapter 10. We introduce the algorithm which in fact is a framework for approximate labeling algorithms and show that it is an FPTAS when used with certain labeling strategies. We also present the running times of two instantiations of the framework. In an extensive computational study, we show the superiority above other FPTASes from the literature.

Chapter 11 finishes the second part. We summarize the findings and pose open problems in the field.

2. Definitions and Notation

In this section, we give necessary definitions and notation. Notation exclusive for the main parts are given in the corresponding chapters.

As naming conventions, sets are denoted with upper-case letters (A), vectors as boldface lower-case letters ($\boldsymbol{v}$), scalars as lower-case Greek or Latin letters (α, a). Algorithms are usually set in calligraphic font ($\mathcal{A}$) and problems similar to sets (P); derived problems are written with the variant description as superscript (P^{DEC}).

2.1. Sets, Functions, Relations and Orders

We first give definitions of the basic number sets in this thesis. With $\mathbb{N}$ we denote the set of natural numbers not including 0, with $\mathbb{Z}$ we denote the set of integer numbers, with $\mathbb{Q}$ we denote the set of rational numbers, and with $\mathbb{R}$ we denote the set of real numbers. With $[n]$ for $n \in \mathbb{N}$, we denote the set $\{1, \dots, n\}$.

A *partition* of a set M is a set of pairwise disjoint subsets $\{A_1, \dots, A_n\}$ of M such that $\bigcup_{i=1}^{n} A_i = M$. A *$k$-partition* of a set M is a partition of M with cardinality k.

For a function $f\colon X \to Y$, we denote for $S \subseteq X$ the *image* of S under f by $f(S) \coloneqq \{f(s) \mid s \in S\}$. The set S_n for $n \in \mathbb{N}$ denotes the set of all bijective functions, or *permutations*, $\pi\colon [n] \to [n]$.

The ith component of a vector $\boldsymbol{a} \in \mathbb{R}^n$ is denoted as $\boldsymbol{a}_i$ for $i \in [n]$. We use the same notation also for the ith element of a tuple or sequence. By $\mathbf{0}^n \in \mathbb{R}^n$ we denote the vector of all 0 and by $\mathbf{1}^n \in \mathbb{R}^n$ we denote the vector of all 1. If the number of components is clear from the context, we simply write $\mathbf{1}$ or $\mathbf{0}$. For vectors $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{R}^n$, we denote with $\leq$ the component-wise less-or-equal order, i.e., $\boldsymbol{v} \leq \boldsymbol{w} :\Longleftrightarrow \boldsymbol{v}_i \leq \boldsymbol{w}_i$, for all $i \in [n]$. For $M \subseteq \mathbb{R}^n$ with $M_>$ we denote the subset of M with vectors greater than $\mathbf{0}$, i.e., $M_> \coloneqq \{\boldsymbol{x} \in M \mid \boldsymbol{x} > \mathbf{0}\}$ and with $M_\geq$, we denote the subset of M with vectors at least $\mathbf{0}$, i.e., $M_\geq \coloneqq \{\boldsymbol{x} \in M \mid \boldsymbol{x} \geq \mathbf{0}\}$.

In this work, we override the traditional meaning of max and min in the sense of greatest and least element. Instead, we use these operators in the sense of maximal and minimal elements in a set. Formally, we define $\max S \coloneqq \{x \in S \mid \forall x' \in S \backslash \{x\} : x' \not\geq x\}$ and $\min S \coloneqq \{x \in S \mid \forall x' \in S \backslash \{x\} : x' \not\leq x\}$. If $\max M$ is just a singleton $\{m\}$, we abuse the notation a bit and write $\max M = m$, to be compatible with the usual notation.

We denote the domination relation by $\boldsymbol{v} \preceq \boldsymbol{w}$ as a strict version of $\leq$, i.e., $\boldsymbol{v} \preceq \boldsymbol{w} :\Longleftrightarrow \boldsymbol{v} \leq \boldsymbol{w}$ and $\boldsymbol{v} \neq \boldsymbol{w}$. As a characterization, it follows that $\boldsymbol{v} \preceq \boldsymbol{w} \iff \forall i \in [n] : \boldsymbol{v}_i \leq \boldsymbol{w}_i$ and $\exists i \in [n] : \boldsymbol{v}_i < \boldsymbol{w}_i$. The $\succeq$-relation is defined analogously. We say a vector $\boldsymbol{v} \in \mathbb{R}^n$ is *nondominated* in a set $M \subseteq \mathbb{R}^n$ if there is no $\boldsymbol{w} \in M$ with

$\boldsymbol{w} \preceq \boldsymbol{v}$. Moreover, we say that $\boldsymbol{v}$ is *weakly nondominated* in M if there is no $\boldsymbol{w} \in M$ with $\boldsymbol{w} < \boldsymbol{v}$ (component-wise). The subset of weakly nondominated elements in a set $M \subseteq \mathbb{R}^n$ is denoted by $\operatorname{wmin} M$.

We denote by $\leq_{\text{lex}}$ the lexicographic less-or-equal order. The lexicographic minimum of a set $M \subseteq \mathbb{R}^n$ is denoted by $\operatorname{lexmin} M$. A maximization version of all these terms is defined analogously.

For a function $f\colon \mathbb{N}^n \to \mathbb{N}$, we define the set $\mathcal{O}(f) \coloneqq \{g\colon \mathbb{N}^n \to \mathbb{N} \mid \exists \boldsymbol{k}' \in \mathbb{N}^n, c > 0\ \forall \boldsymbol{k} \geq \boldsymbol{k}'\colon g(\boldsymbol{k}) \leq cf(\boldsymbol{k})\}$. After D. Knuth [88], we define $f \in \Omega(g) :\iff g \in \mathcal{O}(f)$ and $f \in \Theta(g) :\iff (f \in \mathcal{O}(g) \wedge f \in \Omega(g))$. For $\boldsymbol{x} \in \mathbb{R}^n$, we denote the set of functions asymptotically growing at most polynomially in $\boldsymbol{x}$ by $\operatorname{poly}(\boldsymbol{x}) \coloneqq \bigcup_{k\in\mathbb{N}} \mathcal{O}(\boldsymbol{x}_1^k + \cdots + \boldsymbol{x}_n^k)$.

We denote the Minkowski sum of two sets $A, B \subseteq \mathbb{R}^n$ by $A + B \coloneqq \{a + b \mid a \in A, b \in B\}$.

2.2. Linear Algebra, Topology, and Polyhedra

Since in multiobjective optimization the value of a solution is a vector, linear algebra is a central building block. For a set of vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k \in \mathbb{R}^n$ and a set of scalars $a_1, \ldots, a_k \in \mathbb{R}$ the vector $\sum_{i=1}^k a_i \boldsymbol{v}_i$ is called a *linear combination* of $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k \in \mathbb{R}^n$. If for a linear combination of $v_1, \ldots, v_k$ it holds that $\sum_{i=1}^k a_i = 1$ then the vector $\sum_{i=1}^k a_i \boldsymbol{v}_i$ is called an *affine combination* of $v_1, \ldots, v_k$. A linear combination $\sum_{i=1}^k a_i \boldsymbol{v}_i$ of $v_1, \ldots, v_k$ is called a *conical combination* of $v_1, \ldots, v_k$ if $a_i \geq 0$ for all $i \in [n]$. An affine combination of $v_1, \ldots, v_k$ which is also a conical combination of $v_1, \ldots, v_k$ is called a *convex combination*.

For two vectors $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{R}^n$, the product $\boldsymbol{v} \otimes_{\text{c}} \boldsymbol{w}$ denotes the component-wise vector product

$$\boldsymbol{v} \otimes_{\text{c}} \boldsymbol{w} \coloneqq \begin{pmatrix} \boldsymbol{v}_1 \boldsymbol{w}_1 \\ \vdots \\ \boldsymbol{v}_n \boldsymbol{w}_n \end{pmatrix}.$$

The standard inner product of $\boldsymbol{v}$ and $\boldsymbol{w}$ is written as a matrix product $\boldsymbol{v}^T \boldsymbol{w}$. We define for $\boldsymbol{x} \in \mathbb{R}^n$ and $y \in \mathbb{R}$ the term $\boldsymbol{x}^y$ as the vector of the component-wise exponential function $(\boldsymbol{x}_i^y)_{i\in[n]}$.

We denote the ith row vector of a matrix $A \in \mathbb{R}^{m\times n}$ by $\boldsymbol{a}_i$ and the ith column vector by $\boldsymbol{A}_i$. The element in the ith row and jth column is written as a_{ij}. The ith *unit vector* in $\mathbb{R}^n$ for $i \in [n]$ is the vector $\boldsymbol{e}_i$ with $(\boldsymbol{e}_i)_j = 0$ for $j \in [n]\backslash\{i\}$ and $(\boldsymbol{e}_i)_i = 1$. We denote the *rank* of a matrix $A \in \mathbb{R}^{m\times n}$ by $\operatorname{rank}(A)$. Similar to the Minkowski sum, for a set $M \subseteq \mathbb{R}^m$ and a matrix $A \in \mathbb{R}^{m\times n}$, we define $AM \coloneqq A \cdot M \coloneqq \{A\boldsymbol{x} \mid \boldsymbol{x} \in M\}$.

We also introduce a few definitions from topology to define terms such as interior and boundary. For $p \geq 1$, an *ℓ_p-norm* is a function $\|\cdot\|_p : \mathbb{R}^n \to \mathbb{R}_{\geq}$ with $\|\boldsymbol{x}\|_p \coloneqq (\sum_{i\in[n]} \boldsymbol{x}_i^p)^{\frac{1}{p}}$. The term $\|\boldsymbol{x}\|_\infty$ for a vector $\boldsymbol{x} \in \mathbb{R}^n$ denotes the *maximum norm* or *ℓ_∞-norm*: $\|\boldsymbol{x}\|_\infty \coloneqq \max_{i\in[n]} |\boldsymbol{x}_i|$. An (open) *$d$-ball* centered in $\boldsymbol{x} \in \mathbb{R}^n$ is the set $B_d(\boldsymbol{x}) \coloneqq \{\boldsymbol{y} \in \mathbb{R}^n \mid \|\boldsymbol{x} - \boldsymbol{y}\|_2 < d\}$. The norm $\|\cdot\|$ for topological terms is of

no special interest in this thesis, so we can just assume it is the Euclidean norm. The complement of a subset M of $\mathbb{R}^n$ is $M^c := \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{x} \notin M\}$. For a set $M \subseteq \mathbb{R}^n$ we define the *boundary* of M, or ∂M, as the set of all points $\boldsymbol{x}$ such that for every d the d-ball centered in $\boldsymbol{x}$ contains points in M and not in M or $\partial M := \{x \in \mathbb{R}^n \mid \forall d \in \mathbb{R} : B_d(\boldsymbol{x}) \cap M \neq \emptyset \text{ and } B_d(\boldsymbol{x}) \cap M^c \neq \emptyset\}$. A vector $\boldsymbol{x} \in M$ is in the *interior* of M, denoted by $\boldsymbol{x} \in \operatorname{int} M$ if there exists a $d > 0$ such that $B_d(\boldsymbol{x}) \subseteq M$.

The *convex hull* (*conical hull*, *affine hull*) $\operatorname{conv} M$ ($\operatorname{cone} M$, $\operatorname{aff} M$) of a set $M \subseteq \mathbb{R}^n$ is the set of all convex (conical, affine) combinations of vectors in M. The vector $\boldsymbol{x}$ is said to be in the *relative interior* of M, or $\boldsymbol{x} \in \operatorname{ri} M$ if there is a $d > 0$ such that $B_d(\boldsymbol{x}) \cap \operatorname{aff} M \subseteq M$.

For $\boldsymbol{a} \in \mathbb{R}^n$ and $\alpha \in \mathbb{R}$, a *half-space* is the set $\{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{a}^T\boldsymbol{x} \leq \alpha\}$. An n-polyhedron is the intersection of finitely many half-spaces in $\mathbb{R}^n$. The *dimension* of an n-polyhedron P is $\dim P = n - \max\{\operatorname{rank}(A) \mid A \in \mathbb{R}^{n\times n} \text{ and for every } \boldsymbol{x}, \boldsymbol{y} \in P\colon A\boldsymbol{x} = A\boldsymbol{y}\}$. For a vector $\boldsymbol{c} \in \mathbb{R}^n$ and $\alpha := \min\{\boldsymbol{c}^T\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{x} \in P\}$, we call the set $\{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{c}^T\boldsymbol{x} = \alpha\}$ a *supporting hyperplane of* P if α exists. A *face* of an n-polyhedron P is either P itself or the intersection of P with a supporting hyperplane. Observe that faces are polyhedra themselves. A *facet* is a face of dimension $\dim P - 1$. Faces of dimension 0 are called *extreme points* of P. We denote the set of extreme points by $\operatorname{vert} P$. For an n-polyhedron P we call $P^I := \{x \in \mathbb{R}^n \mid x \in \operatorname{conv}(P \cap \mathbb{Z}^n)\}$ the *integer hull* of P. A polyhedron with one extreme point is called a *cone*. We call a face of dimension 1 of a cone C an *extreme ray* of C.

A vector $\boldsymbol{v} \in \mathbb{R}^n$ is called a *recession direction* of an n-polyhedron P if there exists an $\boldsymbol{x} \in P$ such that $\boldsymbol{x} + \alpha\boldsymbol{v} \in P$ for every $\alpha \geq 1$. The *recession cone* of an n-polyhedron P, denoted by $\operatorname{rec} P$, is the set of all its recession directions. An extreme ray of $\operatorname{rec} P$ is called an *extreme recession direction* of P.

We also need a few terms from convex analysis and take the definitions from Rockafellar and Wets [116]. For an n-polyhedron P we denote the *tangent cone* of P in a vector $\boldsymbol{y} \in P$ by $T_P(\boldsymbol{y}) := \{\boldsymbol{x} \in \mathbb{R}^d \mid \text{there is } \lambda > 0 \text{ with } \boldsymbol{y} + \lambda\boldsymbol{x} \in P\}$. The *normal cone* of an n-polyhedron P in a vector $\boldsymbol{x} \in P$ is the set $N_P(\boldsymbol{x}) := \{\boldsymbol{a} \in \mathbb{R}^d \mid \boldsymbol{a}^T\boldsymbol{x}' \leq \boldsymbol{a}^T\boldsymbol{x}, \boldsymbol{x}' \in P\}$. The *polar dual* of a cone $C \subseteq \mathbb{R}^n$ is the set $C^* = \{\boldsymbol{y} \in \mathbb{R}^n \mid \boldsymbol{y}^T\boldsymbol{x} \leq 0, \boldsymbol{x} \in P\}$.

2.3. Graphs

Now we turn to definitions from graph theory.

Definition 2.1 ((Undirected) Graph)**.** An *(undirected) graph* is a pair (V, E) where V is a finite set and $E \subseteq \{\{u, w\} \mid u, w \in V, u \neq w\}$. The elements of V are called *nodes* and the elements of E are called *edges*.

Definition 2.2 (Directed Graph)**.** A *directed graph* or *digraph* is a pair (V, A) where V is a finite set and $A \subseteq V \times V$. The elements of V are called *nodes* and the elements of A are called *arcs*.

We say for a graph $G = (V, E)$ that two nodes $v, w \in V$ are *adjacent* in G if there is an edge $\{v, w\} \in E$. A node $v \in V$ and an edge $e \in E$ are *incident* in G if $v \in e$. In a directed graph $G = (V, A)$, a node v is a successor of w and w is the predecessor of v if $(w, v) \in A$. An arc $(v, w) \in A$ is called out-arc of v and in-arc of w.

A *(directed) path* in a graph (V, E) (directed graph (V, A)) is a sequence of nodes from V : $(v_1, v_2, \ldots, v_{l-1}, v_l)$ where for each two consecutive nodes v_i and v_{i+1} the edge $\{v_i, v_{i+1}\} \in E$ (the arc $(v_i, v_{i+1}) \in A$) for $i \in [l-1]$. We say there is a (directed) path from node v_1 to node v_l, which is symmetric in the undirected case. The edge $e = \{v_i, v_{i+1}\} \in E$ (arc $a = (v_i, v_{i+1}) \in A$) is said to be *on the (directed) path* p. The *length* of a (directed) path is the number of edges (arcs) on the path, i.e., $l - 1$ in the above cases. A *simple (directed) path* is a (directed) path for which every node on the path is incident to at most two edges on the path (every node has at most one in- and one out-arc). The set of all directed simple paths from node s to node t in G is denoted by $\mathcal{P}_{s,t}^G$. A (directed) *cycle* in a graph G is a (directed) path $(v_1, v_2, \ldots, v_{l-1}, v_l)$ in G with $v_1 = v_l$. A *Hamiltonian cycle* in a graph is a simple cycle of size n.

A *subgraph* of a graph $G = (V, E)$ is a graph $G' = (V', E')$ with $V' \subseteq V$ and $E' \subseteq E$. A *forest* is a graph in which no cycle exists. We call a graph (directed graph) *connected* (*strongly connected*) if there is a path (directed path) from every node to all other nodes. A connected forest is called a *tree*. A *subtree* of a graph G is a subgraph of G which is a tree. A subgraph $G' = (V', E')$ of a graph $G = (V, E)$ *spans* G if $V \subseteq V'$ holds. A spanning subtree of a graph is also called *spanning tree*. The set of spanning trees of a graph G is denoted as $\mathcal{T}_G$.

Let $G = (V, E)$ be an undirected graph. A *cut* in G induced by a 2-partition $\{S, S'\}$ of V is the set of edges with one end in S and one in S'. The graph G is said to be *bipartite* if there is a 2-partition (or bipartition) $\{S, S'\}$ of V such that E is exactly the cut induced by $\{S, S'\}$. A graph G is said to be *complete* if $E = \{\{v, w\} \mid v, w \in V, v \neq w\}$.

2.4. Computational Complexity and Encoding

When considering the complexity of computational problems, the encoding of inputs and outputs needs to be formalized. This is done by the use of alphabets, strings and languages. An *alphabet* is a finite set. Throughout this thesis, we can assume alphabets to be the set $\{0, 1\}$. A *string* (or word) over an alphabet Σ is a sequence of elements from the alphabet, e.g., $(0, 1, 0, 0, 0, 1, 1, 1)$. To make writing strings easier, we often simply write 01000111 instead. For a string x, we denote by $|x|$ the length of the string which is the length of the sequence. Finally, a *language* is a set of strings over a given alphabet, e.g., $\{0, 1, 01, 11, 100, 101, 110, \ldots\}$.

To make defining languages easier, we make use of the *star-operator* (*): For a given alphabet Σ, the term Σ^* is the language of all possible strings over the alphabet Σ.

As we discuss linear programming and thus continuous problems and deal with encoding lengths of numbers, we follow Grötschel, Lovász, and Schrijver [70] in defining these notions. A natural number $n \in \mathbb{N}$ can be encoded by a string over

$\{0,1\}$ in binary representation. Thus, an upper bound for representing n is $\log n$ and we define $\langle n \rangle \coloneqq \lceil \log(n+1) \rceil$. For an integer number $z \in \mathbb{Z} \backslash \{0\}$, we add a sign-bit and define $\langle z \rangle \coloneqq 1 + \lceil \log(|z|+1) \rceil$. To represent 0, a string of size 1 is sufficient in both cases. Further, we define the encoding length of a rational number $r \in \mathbb{Q}$ with $r = \frac{p}{q}$ and $p \in \mathbb{Z} \backslash \{0\}$, $q \in \mathbb{N}$ such that $|p|$ and q are co-prime, as $\langle r \rangle \coloneqq 1 + \lceil \log(|p|+1) \rceil + \lceil \log(q+1) \rceil$. For a vector $\boldsymbol{x} \in \mathbb{Q}^n$, $\langle \boldsymbol{x} \rangle \coloneqq \sum_{i \in [n]} \langle \boldsymbol{x}_i \rangle$ and for a matrix $A \in \mathbb{Q}^{m \times n}$ it is $\langle A \rangle \coloneqq \sum_{i \in [n]} \sum_{j \in [m]} \langle a_{ji} \rangle$. Observe that $\langle A \rangle \in \Omega(mn)$ and $\langle \boldsymbol{x} \rangle \in \Omega(n)$.

For our study of the complexity of problems the following working definition of computational problems suffice for this thesis: A *computational problem* is a problem suitable to be solved by a computer. Before we define the problems with which we are concerned in this thesis, we first define the formal meaning of solving by a computer.

Computational Models. The complexity of a computational problem is always defined with respect to a certain computational (or machine) model. In classic complexity theory, this is usually the Turing machine. Since we do not construct Turing machines nor use special properties which require to define a notation, we refer the reader to the book by Arora and Barak [8] for its formal definition.

It is tedious to implement algorithms on Turing machines to find polynomial upper bounds on the running time. Thus, we usually implement algorithms on variants of the *random access machine* (RAM). Since a Turing machine can simulate a RAM with at most polynomial overhead, a polynomial bound on the running time of a RAM implies the existence of a polynomial-time constrained Turing machine. A very concise definition of RAMs and worst-case running time is given by Cormen et al. [36].

We stress here that in Cormen et al. [36] registers are allowed to store integer numbers bounded by $|x|^k$ for some fixed $k \in \mathbb{N}$ on each input x. In other words: The encoding length of a number is restricted by $k \log |x|$. One can prove that in this case the Turing machines can still simulate these RAMs with polynomial overhead (as opposed to a 0/1-RAM).

The RAM also plays a crucial role in enumeration complexity which is investigated in more depth in Chapter 3.

Decision Problems. Informally, a *decision problem* is a computational problem, where the only possible answers are "yes" or "no". Formally, a decision problem L is a subset of a base set Ω. We call the elements of L the *yes-instances* of L and the elements in $\Omega \backslash L$ the *no-instances* of L. The *complement* of a decision problem L is the set co-$L \coloneqq \Omega \backslash L$.

In this thesis, the base set usually is the set $\{0,1\}^*$. To make descriptions less tedious, we implicitly use a binary encoding of the objects we work with. For example, the problem to decide whether a graph G contains a Hamiltonian cycle can be stated as the following set: $\{\text{Graph } G \mid G \text{ contains a Hamiltonian cycle}\}$. Thus, the base set is the set of all graphs in a binary encoding. The representation of graphs as strings over $\{0,1\}$ can be crucial, but is not of special interest in this thesis.

We say a decision problem L is *decidable* if there is a Turing machine which terminates on every instance of L and returns "yes" on every yes-instance of L and "no" on every no-instance of L.

Optimization Problems. Optimization problems are the main concern in this thesis, especially multiobjective optimization problems. In this section, we introduce the classic single-objective optimization notion. Because maximization is defined analogously to minimization, we introduce all terms for minimization and assume the maximization analogon to be defined accordingly.

Since this thesis lies between mathematics and theoretical computer science, it is necessary to clarify the definitions used, since they often differ in subtle ways. In mathematical optimization, optimization problems are usually defined in the following way:

Definition 2.3 (Mathematical Optimization Problem)**.** A *mathematical optimization problem* consists of a set of instances. An *instance* of an optimization problem consists of a set of *solutions* $\mathcal{X}$ and an *objective function* $f\colon \mathcal{X} \to \mathbb{R}$. The goal is to find an $x \in \mathcal{X}$ such that there is no $y \in \mathcal{X}$ with $f(y) \leq f(x)$.

But the above definition of optimization problems does not consider input encodings. When we study the efficiency of algorithms and want to find the fastest algorithm for a problem at hand, it is crucial to consider how the input to our algorithm is given. This is less relevant if we are only interested in the fact if an algorithm runs in polynomial time or not. In this case, we can assume that the conversion of input formats can be performed in polynomial time and we can concentrate on solving the problem itself. Moreover, we do not use real but rational numbers in our inputs because our computational models do not handle real numbers. Hence, we deal with a more algorithmically centered definition of optimization problems based on the definition by Papadimitriou and Steiglitz [110] and by Korte and Vygen [89].

Definition 2.4 (Algorithmic Optimization Problem)**.** An algorithmic optimization problem (I, S, v) consists of

- a set of *instances* $I \subseteq \Sigma^*$,
- a mapping $S\colon I \to 2^{\Sigma^*}$ which maps an instance to its set of *solutions*, and
- a mapping $v\colon I \times S(I) \to \mathbb{Q}$ which maps each solution of a given instance to its *value* in this instance.

For every $x \in I$ there must exist a polynomial $p\colon \mathbb{N} \to \mathbb{N}$ such that for every $s \in S(x)$ we have $|s| \in \mathcal{O}(p(|x|))$. We assume that S is computable, $S(x)$ is polynomial-time decidable for every $x \in I$, and v is polynomial-time computable. The goal is to find for an instance $x \in I$ a solution $s \in S(x)$ such that there is no $s' \in S(x)$ with $v(x, s') \leq v(x, s)$.

In the following, we mainly use the definition of algorithmic optimization problems. An optimization problem is a *combinatorial optimization problem* if there exists a polynomial $p\colon \mathbb{N} \to \mathbb{N}$ such that for every instance x the set $S(x)$ is a subset of $\{0,1\}^{p(|x|)}$ ($\mathcal{X} \coloneqq \{0,1\}^n$). Since we only deal with combinatorial optimization problems with linear objective functions, we also assume that for every instance $x \in I$ there exists a vector $\boldsymbol{c}_x \in \mathbb{Q}^{p(|x|)}$ such that we can write $v(x, \boldsymbol{s}) = \boldsymbol{c}_x^T \boldsymbol{s}$.

Note also that we are defining the optimality notion in the sense of minimal elements and not of a least element. This does not make a difference here but is easier to generalize to multiobjective optimization. Further, we are not concerned with problems with topologically open objective function images, hence we do not have to deal with infimum notions.

For an optimization problem P, an instance $x \in I$ and a solution $s \in S(x)$, we sometimes write $v(s)$ instead of $v(x, s)$ if the instance is clear from context.

For every (algorithmic) optimization problem, there is an associated decision problem which is crucial for the study of the computational complexity of optimization problems:

Definition 2.5 (Canonical Decision Problem)**.** For an algorithmic optimization problem $O = (I, S, v)$, the corresponding *canonical decision problem* O^{DEC} is the following set:

$$\{(x, k) \in I \times \mathbb{Q} \mid \exists s \in S(x)\colon v(x, s) \leq k\}$$

In other words, a pair (x, k) is a yes-instance of the canonical decision problem iff x represents a valid instance and there exists a feasible solution $s \in S(x)$ such that its value $v(x, s)$ is at most k. One of the essential observations by the work of Karp, Cook and others in the 1970s was that if for any optimization problem O if O^{DEC} is **NP**-hard, so is O (cf., e.g., Garey and Johnson [69]).

Complexity Classes. The study of the complexity of computational problems usually involves assigning problems to a complexity class. The most important of these classes is the class **P**, which is often regarded as being the class of efficiently solvable problems.

Definition 2.6 (Polynomial Time Solvable Problems (**P**))**.** The set **P** denotes the set of computational problems which can be solved by a Turing machine in polynomial worst case running-time.

For an optimization problem (I, S, v), we say that an algorithm $\mathcal{A}$ runs in *strongly polynomial time* if the running time of $\mathcal{A}$ on an instance $x \in I$ is independent of the objective function v.

For decision problems, it is often easy to verify that an instance is a yes-instance when also a certificate for a yes-decision is given. This leads to the formal definition of the class **NP**.

Definition 2.7 (Nondeterministic Polynomial Time Solvable Problems (**NP**))**.** The set **NP** is the set of decision problems for which there exists a Turing machine $\mathcal{A}$ with polynomial worst-case running-time and a polynomial p such that

- for a yes-instance x, there exists a string $y \in \{0,1\}^{p(|x|)}$ such that $\mathcal{A}$ returns "yes" on input xy and
- for a no-instance x, $\mathcal{A}$ returns "no" on input xy for any $y \in \{0,1\}^{p(|x|)}$.

Note that our definition of the class **P** is not restricted to decision problems which implies that the term $\mathbf{P} = \mathbf{NP}$ is always false. Nevertheless, the famous **P**-**NP** question can still be stated as: Is $\mathbf{NP} \subset \mathbf{P}$? It makes our lives easier to work with a class **P** which includes all polynomial-time solvable problems, including optimization and enumeration problems.

Symmetrically to **NP**-problems to which it is easy to find yes-certificates, the class **co-NP** is the set of decision problems on which it is easy to reach a no-decision if a no-certificate is given.

Definition 2.8 (**co-NP**)**.** The set **co-NP** is the set of decision problems for which there exists a Turing machine $\mathcal{A}$ with polynomial worst-case running-time and a polynomial p such that

- for a no-instance x, there exists a string $y \in \{0,1\}^{p(|x|)}$ such that $\mathcal{A}$ returns "no" on input xy and
- for a yes-instance x, $\mathcal{A}$ returns "yes" on input xy for any $y \in \{0,1\}^{p(|x|)}$.

In other words, a decision problem L is in **co-NP** iff co-L is in **NP**.

Reducibility and Hardness. To show relationships between problems, we use the notion of simulation and reducibility. A reduction is a binary relation $\vdash$ on computational problems. Moreover, reductions are a basic building block to define traditional hardness. Let us now first introduce Karp-Reduction for decision problems:

Definition 2.9 (Polynomial Time Many-One Reduction or Karp-Reduction)**.** A decision problem L' with base set Ω' can be *Karp-reduced* to a decision problem L with base set Ω if there exists a polynomial-time transformation $f\colon \Omega' \to \Omega$ such that $x \in L' \Leftrightarrow f(x) \in L$. We write $L' \leq_P L$.

To be able to also reduce among other problems than decision problems, we use Cook-reduction.

Definition 2.10 (Polynomial Time Turing Reduction or Cook-Reduction)**.** A computational problem P' can be *Cook-reduced* to a computational problem P if there exists a polynomial-time algorithm solving P' which may use an oracle solving P which accounts only for a constant in the running time. We write $P' \leq_T P$.

Now for the notion of $\boldsymbol{C}$-hardness for a given set of computational problems $\boldsymbol{C}$, we have our two basic ingredients: The complexity class $\boldsymbol{C}$ and an often implicit reduction notion $\vdash$.

Definition 2.11 ($\boldsymbol{C}$-hardness)**.** For a complexity class $\boldsymbol{C}$, a problem P is said to be $\boldsymbol{C}$-hard w.r.t. to a reduction $\vdash$ if for every problem $P' \in \boldsymbol{C}$ we have $P' \vdash P$.

For **NP**-hardness, we usually implicitly use Karp-reductions when reducing among decision problems and Cook-reductions if we reduce to other problems, e.g., optimization problems.

A reduction always has a direction in the sense that the complexity transfers from one problem to the other but in general not the other way around. Sometimes we want to show that two problems have an equivalent complexity. We can do this by using a notion of equivalence:

Definition 2.12 (Polynomial-Time Equivalence)**.** Two computational problems P and Q are said to be *polynomial-time equivalent* if $P \leq_T Q$ and $Q \leq_T P$. We write $P =_T Q$. Two decision problems L and L' are said to be *polynomial-time equivalent* if $L \leq_P L'$ and $L' \leq_P L$. We write $L =_P L'$.

2.5. Multiobjective Optimization

Let us now turn to the main kind of problems in this thesis. Without loss of generality, we concentrate on the minimization problem variants. Maximization objectives can be seen as negations of minimization objectives and are used in this regard. We again approach the problems from a computer scientists viewpoint and extend the definition by Papadimitriou and Steiglitz [110], and Korte and Vygen [89] to multiobjective problems.

Definition 2.13 ((Algorithmic) Multiobjective Optimization (MOO) Problem)**.** An algorithmic multiobjective optimization problem (I, S, v) consists of

- a set of *instances* $I \subseteq \Sigma^*$,
- a mapping $S\colon I \to 2^{\Sigma^*}$ which maps an instance to its set of *solutions*, and
- a mapping $v\colon I \times S(I) \to \mathbb{Q}^d$ which maps each solution of an instance to a vector.

It must hold that for every $x \in I$ there exists a polynomial $p\colon \mathbb{N} \to \mathbb{N}$ such that for every $s \in S(x)$ we have $|s| \in \mathcal{O}(p(|x|))$. We again assume that S is computable, $S(x)$ is polynomial-time decidable for every $x \in I$, and v is polynomial-time computable. We consider a MOO instance $x \in I$ to be solved if we output the set $\mathcal{Y}_N \coloneqq \min v(x, S(x)) = \min\{v(x, s) \mid s \in S(x)\}$.

For an instance $x \in I$ of a MOO problem P, we call the set $\mathcal{Y}_N$ the *nondominated set* or *Pareto-front* of x. A solution $s \in S(x)$ is called *efficient* or *Pareto-optimal* if $v(x, s) \in \mathcal{Y}_N$. A solution s is *weakly efficient* or *weakly Pareto-optimal* if $v(x, s) \in \operatorname{wmin}\{v(x, s) \mid s \in S(x)\}$. Analogous to the value of a solution in single-objective optimization, we call $v(x, s) \in \mathbb{Q}^d$, for an instance $x \in I$ and $s \in S(x)$, the *value vector*

of s. Usually, we expect algorithms also to produce an efficient solution $s \in S(x)$ for every $y \in \mathcal{Y}_N$ such that $v(x, s) = y$, though we do not require this formally. Observe that this definition also subsumes single objective optimization problems.

Note that in general the nondominated set is an infinite set and so these problems do not pose examples of computational problems. Hence, we are concerned with a special kind of MOO problem: A *multiobjective combinatorial optimization problem (MOCO)* is a MOO problem (I, S, v) where there exists a polynomial $p\colon \mathbb{N} \to \mathbb{N}$ such that for every instance $x \in I$ the set $S(x)$ is a subset of $\{0,1\}^{p(|x|)}$. Again, we only deal with problems having linear objective functions, so we again assume that for every MOCO instance $x \in I$ there exists a matrix $C_x \in \mathbb{Q}^{d \times p(|x|)}$ such that we can write $v(x, \boldsymbol{s}) = C_x^T \boldsymbol{s}$.

A very important subset of the nondominated set is the set of nondominated extreme points of MOCO problems. Given a MOCO problem (I, S, v), we define the *extreme points of the nondominated set* or *nondominated extreme points* of an instance $x \in I$ to be the set $\mathcal{Y}_X \coloneqq \operatorname{vert} \operatorname{conv} v(x, S(x))$. We denote by O^{YEx} the computational problem of finding the set $\mathcal{Y}_X$ for a given MOCO problem O. (An analogous definition can be stated for multiobjective integer linear optimization problems, but we are not concerned with these problems in detail in this thesis.)

The *ideal point* of an instance $x \in I$ of MOO (I, S, v) is the vector $y^x_{\min} \in \mathbb{R}^d$ with $(y^x_{\min})_i = \min\{v(x, s)_i \mid s \in S(x)\}$. We say that the ideal point exists for the instance x if every $\min\{v(x, s)_i \mid s \in S(x)\}$ exists for $i \in [d]$. The *weighted-sum problem* of a MOO $P = (I, S, v)$ with respect to a vector $\boldsymbol{\ell} \in \mathbb{Q}^d$ is the single-objective optimization problem (I^w, S^w, v^w) with $I^w = \{(x, \boldsymbol{\ell}) \mid x \in I\}$, $S^w = S$ and $v^w((x, \boldsymbol{\ell}), s) = \boldsymbol{\ell}^T v(x, s)$. We denote this problem as $P_1(\boldsymbol{\ell})$.

2.5.1. Problem Definitions

In this section, we define the problems that are investigated in this thesis. We usually define the input and assume a canonical encoding of it. For graph problems, this can be an adjacency matrix also encoding the edge-costs or an adjacency list including the edge costs. The set of solutions is always a subset of the edges and the costs are defined by an edge-cost function, thus the cost of a solution is the sum of the costs of the edges in the solution. Consequently, it is sufficient to define the input—not necessarily the concrete input encoding—and the set of feasible solutions for the following problems. From the definition of multiobjective combinatorial optimization problems then follows the goal to find the nondominated set.

Combinatorial Problems

Definition 2.14 (Multiobjective (global) Minimum Cut (MOMC) Problem)**.** The input is a graph $G = (V, E)$ and an edge-cost function $c\colon E \to \mathbb{Q}^d$. A feasible solution is a cut in G.

Definition 2.15 (Multiobjective Unconstrained Optimization (MUCO) Problem)**.** The input is a matrix $C \in \mathbb{Q}^{d \times n}$ and a feasible solution is an element of $\{0,1\}^n$. The

cost of a feasible solution $x \in \{0,1\}^n$ is the matrix-vector product Cx.

We are concerned mainly with the special case of the *biobjective unconstrained optimization (BUCO)* problem where d is fixed to 2.

Definition 2.16 (Multiobjective Spanning Tree (MOST) Problem)**.** We are given an undirected graph $G = (V, E)$ and an edge-cost function $c\colon E \to \mathbb{Q}^d$. A feasible solution is the set of edges of a spanning tree of G from $\mathcal{T}_G$.

A set of edges $M \subseteq E$ of a graph $G = (V, E)$ is called a *matching*, if for every $e, e' \in M$ with $e \neq e'$ it follows that $e \cap e' = \emptyset$. We call a node $v \in V$ *free* (w.r.t. a matching M) if there is no $e \in M$ with with $v \in e$. A matching M is called *perfect* if there is no free node w.r.t. M in V.

Definition 2.17 (Multiobjective Matching Problem (MO-Matching))**.** Given a graph $G = (V, E)$ and an edge-cost function $c\colon E \to \mathbb{Q}^d$, a feasible solution is a perfect matching in G.

A very special case of the matching problem is the *multiobjective assignment problem*, in which we require the input graph to be bipartite.

Definition 2.18 (Multiobjective Assignment Problem (MOAP))**.** Given a complete bipartite undirected graph $G = (V, E)$ with bipartition $V = U \dot\cup W$ and $|U| = |W|$ and an edge-cost function $c\colon E \to \mathbb{Q}^d$. The feasible solutions of the MOAP problem are all perfect matchings in G.

Definition 2.19 ((Single Pair) Multiobjective Shortest Path (MOSP) Problem)**.** Given a digraph $G = (V, A)$, two nodes $s, t \in V$ and an edge-cost function $c\colon A \to \mathbb{Q}^d$. A feasible solution is the set of edges on a path in $\mathcal{P}_{s,t}^G$. We define $c(p) \coloneqq \sum_{a \in A \text{ on } p} c(a)$ for a path $p \in \mathcal{P}_{s,t}^G$.

Definition 2.20 (Multiobjective Traveling Salesperson (MOTSP) Problem)**.** The input is a complete graph $G = (V, E)$ and an edge-cost function $c\colon E \to \mathbb{Q}^d$. A feasible solution is the set of edges on a Hamiltonian cycle in G. We define $c(C) \coloneqq \sum_{e \in E \text{ on } C} c(e)$ for a Hamiltonian cycle C.

Note that although the above problems are MOCO problems, we did not define them according to the definition of MOCO problems above. We can, however, construct a solution set in $\{0,1\}^{\text{poly}(|x|)}$ and an objective function matrix C_x in a canonical way for the graph problems above: A feasible solution is always a subset of the edges. Thus, for every input graph $G = (V, E)$ (or (V, A) for digraphs), depending on the input encoding, p maps an instance to its number of edges and the feasible solutions are vectors with as many components as G has edges. For an instance $x \in I$, when we fix an arbitrary ordering of the edges in $E : e_1, \ldots, e_{|E|}$, the ith component of a solution $\boldsymbol{s} \in S(x)$ is 1 if e_i is in the subset of edges $\boldsymbol{s}$ represents, or 0 if it is not in the subset $\boldsymbol{s}$ represents. The objective function matrix then is $C_x \in \mathbb{Q}^{d \times |E|}$ where the ith column is the cost of edge e_i.

Multiobjective Linear Programming

We first define a meta-version of the multiobjective linear programming (MOLP) problem, define a few terms and then define the concrete computational problems we consider in this thesis. The following definition thus is not a computational problem in the sense of our definition of a computational problem since the output set is uncountable and inherently unsuitable to be solved by a computer.

Definition 2.21 (Multiobjective Linear Programming (MOLP) Scheme)**.** The input is given as a constraint matrix $A \in \mathbb{Q}^{m \times n}$, a cost matrix $C \in \mathbb{Q}^{d \times n}$ and a vector $\boldsymbol{b} \in \mathbb{Q}^m$. We usually write an MOLP in the mathematical form

$$\begin{aligned} \min\ & C\boldsymbol{x} \\ \text{s.t. } & A\boldsymbol{x} \leq \boldsymbol{b} \\ & \boldsymbol{x} \in \mathbb{R}^n. \end{aligned}$$

We define the *feasible set* as $P \coloneqq \{\boldsymbol{x} \in \mathbb{R}^n \mid A\boldsymbol{x} \leq \boldsymbol{b}\}$ and the *feasible set in objective space* as $\mathcal{Y} \coloneqq C \cdot P \coloneqq \{Cx \mid x \in P\}$. The nondominated set or Pareto-front is the set $\mathcal{Y}_N \coloneqq \min \mathcal{Y}$.

A pair $(\boldsymbol{a}_i, b_i)$ with $\boldsymbol{a}_i$ being the ith row of A is called a *constraint*. For a vector $\boldsymbol{x} \in \mathbb{R}^n$, a constraint $(\boldsymbol{a}_i, b_i)$ is called *active* if $\boldsymbol{a}_i^T \boldsymbol{x} = b_i$. A set of constraints $\{(\boldsymbol{a}^1, b^1), \ldots, (\boldsymbol{a}^k, b^k)\}$ is called *linearly independent* if the set of vectors $\{(\boldsymbol{a}_1^1, \ldots, \boldsymbol{a}_n^1, b^1)^T, \ldots, (\boldsymbol{a}_1^k, \ldots, \boldsymbol{a}_n^k)\}$ is linearly independent. A vector $\boldsymbol{x} \in \mathbb{R}^n$ is called *basic* if there are n linearly independent constraints active in $\boldsymbol{x}$.

Following Heyde and Löhne [75], we define the *upper image* as the polyhedron $\mathcal{P} \coloneqq \mathcal{Y} + \mathbb{R}^d_{\geq}$. It gives us a better description of the nondominated set, since $\mathcal{Y}_N \subseteq \partial\mathcal{P}$.

We define three computational problems as incarnations of the MOLP scheme: MOLP^{XEx}, MOLP^{YEx} and MOLP^{YFa}. Let in the following a MOLP consisting of $A \in \mathbb{Q}^{m \times n}$, $C \in \mathbb{Q}^{d \times n}$, and $\boldsymbol{b} \in \mathbb{Q}^m$ be given. In MOLP^{XEx} we are interested in the Pareto-optimal basic feasible solutions:

Definition 2.22 (MOLP^{XEx})**.** The input is a MOLP as above and the output is the set $\{\boldsymbol{x} \in \mathbb{R}^n \mid Cx \in \mathcal{Y}_N, \boldsymbol{x} \text{ is basic}, A\boldsymbol{x} \leq \boldsymbol{b}\}$.

Analogously to single objective optimization, we are often more interested in the optimal value and one representative solution. This view is covered by the problem MOLP^{YEx}.

Definition 2.23 (MOLP^{YEx})**.** The input is a MOLP as above and the output is the set $\operatorname{vert} \mathcal{P}$.

Let us note that MOLP^{XEx} and MOLP^{YEx} are different problems. We can have instances of MOLP^{XEx} with an exponential output, where MOLP^{YEx} has an output size of 1.

In some applications as in finance, it can be more interesting to have a facet representation of the upper image, which is covered by the problem MOLP^{YFa}.

Definition 2.24 (MOLP^{YFA})**.** The input is a MOLP as above and the output is a set of vectors $\{\boldsymbol{a}_1, \dots, \boldsymbol{a}_k\} \subseteq \mathbb{R}^d$ such that $\mathcal{P} = \{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{a}_i^T \boldsymbol{x} \geq 1 \text{ for all } i \in [k]\} + \mathbb{R}^d_{\geq}$.

A linear programming problem is a MOLP^{YEX} with $d = 1$.

Other Problems

Definition 2.25 (Multiobjective Integer Minimum Cost Flow Problem (MO-Z-Flow))**.** The input is

- a digraph $G = (V, A)$,
- two nodes $s, t \in V$,
- a *required flow* $R \in \mathbb{Z}_{\geq}$,
- a *capacity function* $b\colon A \to \mathbb{Z}_{\geq}$, and
- an edge-cost function $c\colon A \to \mathbb{Q}^d_{\geq}$.

A feasible solution is a function $f\colon A \to \mathbb{Z}_{\geq}$ such that the following flow conservation constraints hold for every node $v \in V$:

$$\sum_{a=(v,\cdot)\in A} f(a) - \sum_{a=(\cdot,v)\in A} f(a) = \begin{cases} R & v = s \\ -R & v = t \\ 0 & \text{else} \end{cases}$$

and the following capacity constraints hold for every arc $a \in A$:

$$f(a) \leq b(a)$$

The cost of a solution then is defined as $c(f) \coloneqq \sum_{a\in A} f(a)c(a)$. The set of all feasible flows is called $\mathcal{F}$ and the goal is to find the set $\mathcal{Y}_N \coloneqq \min c(\mathcal{F})$.

Note that a canonical encoding for the capacity function in the input is a vector from $\mathbb{Z}^{|A|}_{\geq}$, for the edge-cost function a canonical encoding is a matrix from $\mathbb{Q}^{d\times|A|}$, and also the canonical encoding of a solution is a vector from $\mathbb{Z}^n_{\geq}$. Observe that the MOSP problem is a special case with $R = 1$ and $b(a) = 1$ for all $a \in A$.

A very similar problem is the *Multiobjective Rational Minimum Cost Flow (MO-Flow) Problem* which is essentially the same problem, but a feasible solution is a function $f\colon A \to \mathbb{Q}_{\geq}$. Note that it is neither a specialization nor a generalization of MO-Z-Flow, because MO-Flow is a special case of a MOLP and thus the nondominated set is usually an uncountable set while the nondominated set of the MO-Z-Flow problem is always countable.

Part I.

Output-Sensitive Complexity of Multiobjective Optimization

3. Introduction

In this first part, we investigate the computational complexity of multiobjective optimization problems. In Table 3.1, we summarize the complexity states of the problems investigated in this thesis. In terms of classic complexity theory, only the first column of Table 3.1 was known, excluding most of the **NP**-hardness results. Since every problem could only be classified as intractable, the picture of the complexity landscape of these problems occurred to be entirely black with no shades.

In this situation, it is not clear what constitutes a good algorithm for a multiobjective optimization problem. An optimal algorithm could be an algorithm spending exponential time on every single instance, with the rationale that there are instances where this time is required. Of course, in a practical problem having—for whatever reason—only a small number of nondominated points, this is unsatisfactory. Informally, what we want to design are algorithms that are fast if the nondominated set is small and not too slow if it is large.

In this regard, what should a good theory of computational complexity of multiobjective optimization problems accomplish? The most important aspect is the classification into "easy" and "hard" problems. But the reason to study which problems are easy or hard is to better decide which algorithms to use. If we are constructing an algorithm that would lead to solving the Traveling Salesperson Problem in polynomial time, we should probably check if this algorithm or its analysis is correct (as long as we believe in $\mathbf{P} \neq \mathbf{NP}$). Thus, the ultimate goal is a practical one: To support algorithm designers in coming up with better algorithms.

3.1. Previous Work

The most important property from a computational complexity viewpoint of multiobjective combinatorial optimization (MOCO) problems in their general form is that they are intractable in the sense of the Cobham–Edmonds thesis [33, 53]. For the multiobjective shortest path (MOSP) problem this was proven by P. Hansen in 1979 [74] and the intractability transfers to the general MOCO problem, the matching problem and the integer min-cost flow problem as generalizations. For fractional min-cost flow, intractability (i.e., the number of representative points grows superpolynomially) was proven by G. Ruhe in 1988 [119] and this transfers to the finding of the nondominated extreme points and facets in multiobjective linear programming (MOLP^{YEx} and MOLP^{YFa}) as well as finding Pareto-optimal basic feasible solutions (MOLP^{XEx}). For the multiobjective traveling salesperson (MOTSP) problem, V. A. Emelichev and V. A. Perepelitsa proved intractability in 1992 [60]. The intractability of the multiobjective spanning tree (MOST) problem was proven by H. W. Hamacher

Problem Name	Classic Complexity	Enumeration	
		Nondominated Set	Extreme Points
MOCO	intractable / **NP**-hard	$\notin$ **TotalP**	$\notin$ **TotalP**
MO-TSP	intractable [60] / **NP**-hard [69]	$\notin$ **TotalP**	$\notin$ **TotalP**
MOLP^{XEx}	intractable [127]	$\notin$ **TotalP**	not applicable
MOLP^{YEx}	intractable [119]	**IncP** *	not applicable
MOLP^{YFA}	intractable [119]	**IncP** *	not applicable
MO-Matching	intractable / **NP**-hard	$\notin$ **TotalP**	**TotalP**
MO-Z-Flow	intractable [119] / **NP**-hard	$\notin$ **TotalP**	**IncP**
MOAP	intractable [55] / ~~**NP**-hard [124]~~	**BUCO**-hard	**IncP**
MOSP	intractable [74] / **NP**-hard	$\notin$ **TotalP**	**P**
MOST	intractable [72] / ~~**NP**-hard [124]~~	**BUCO**-hard	**P**
BUCO	intractable [55] / ~~**NP**-hard [124]~~	?	**P**
MOMC	?	**IncP**	**P**

Table 3.1.: State of well-known MOP problems. The positive results all assume that the number of objectives is fixed. The negatives are also true for a small fixed number of objectives. The "$\notin$ **TotalP**"-results are true under the assumption that $\mathbf{P} \neq \mathbf{NP}$.
* Certain assumptions need to be fulfilled.

and G. Ruhe in 1994 [72]. For the multiobjective assignment (MOAP) problem and biobjective unconstrained combinatorial optimization problem, we refer to the book by M. Ehrgott for proofs of intractability [55].

In the case of the MOMC problem, for $d = 2$ it is known that there are $\mathcal{O}(|V|^7)$ nondominated points [3]. For $d > 2$, the number of nondominated points is not known.

Another property which is important in algorithm design in single objective optimization is **NP**-hardness. M. Ehrgott states in [54], that if a single objective problem is **NP**-hard, so is its multiobjective extension. We refer the reader to the paper for exact definitions and proofs. One example problem in this regard is the MO-TSP problem, which thus is **NP**-hard and this transfers to the general MOCO problem, since it is a generalization. In Section 4.1 we investigate **NP**-hardness in multiobjective optimization in more detail.

Further, there exist results on the **NP**- and $\#\boldsymbol{P}$-hardness of decision problems associated with multiobjective optimization problems. While these results are of academic interest, they do not seem to have implications to the multiobjective optimization problems themselves, thus we are not concerned with them in this thesis.

There is one paper about output-sensitive complexity of multiobjective optimization problems by Y. Okamoto and T. Uno from 2010 [104]. They propose an algorithm for the MOST problem, however, the model considered by Okamoto and Uno is different from the one considered in this thesis. They consider enumerating all *supported weakly Pareto-optimal* spanning trees, i.e., for the input graph G all $T \in \mathcal{T}_G$ such that T

is an optimal solution to $P_1(\ell)$ for some $\ell \geq 0$. This includes non-Pareto-optimal spanning trees for $\ell \in \{e_i \mid i \in [d]\}$ and also equivalent trees T and T' with $T \neq T'$ and $c(T) = c(T')$. For this problem, the authors propose an algorithm which runs with polynomial delay by employing a variant of the reverse search method by Avis and Fukuda [9]. The authors also generalize this algorithm to the case of enumerating Pareto-optimal basic feasible solutions of an MOLP if it is nondegenerate, i.e., every basic solution has exactly n active inequalities.

In contrast to the problems investigated in this thesis, there exist many multiobjective optimization problems for which it is known that the nondominated set is only of polynomial size. Figueira et al. [63] investigated many of these problems as special cases of the above problems or entirely new problems and proved polynomial time solvability or **NP**-hardness.

There is also another complexity measure which was applied to multiobjective optimization, i.e., Smoothed Analysis, which was introduced to MOP by Ackermann et al. [1]. We dedicate Section 4.3 to summarize the findings and connection to output-sensitive complexity.

3.2. Contributions and Organization

In the beginning of this part of this thesis, we are concerned with the first column of Table 3.1. The intractability of all the problems given there was long known, but for most of these problems it was assumed that they are **NP**-hard, especially the MOSP, MOAP and MOST problems. We shed some light on the classic theory of **NP**-hardness in multiobjective optimization in Section 4.1. This gives reasons for striking the **NP**-hardness results for the MOAP, MOST and MOSP problems. We further introduce output-sensitive complexity and see the first problem which can be classified as "easy", which is the MOMC problem.

Following in this first part, in Chapter 5 we are concerned with multiobjective linear programming. We show that under mild assumptions the multiobjective linear programming problem also belongs to the easy problems in output-sensitive complexity. This is especially useful, because we can apply these results to MOCO problems to find the complete set of nondominated extreme points. More specific, we prove that for any MOCO problem if we are able to solve certain oracle problems in polynomial time, it is easy in the sense of output-sensitive complexity to find the set of nondominated extreme points. This leads to the complete last column of Table 3.1 and to the positive results for the MOLP problems.

In Chapter 6 we are concerned with showing what cannot be done efficiently in the model of output-sensitive complexity. We demonstrate a general methodology how to show that a multiobjective optimization problem is a hard enumeration problem and use standard assumptions from complexity theory as $\mathbf{P} \neq \mathbf{NP}$. Subsequently, we exemplify this on the multiobjective shortest path problem and show that there is no output-sensitive algorithm for MOSP unless $\mathbf{P} = \mathbf{NP}$. This also leads us to the hardness results for the multiobjective versions of the matching and the integer

min-cost flow problems. Moreover, we introduce an interesting problem that seems to hold us back from getting better algorithms for the multiobjective assignment and multiobjective spanning tree problems and introduce the concept of **BUCO**-hardness. This can be found in the second last column in Table 3.1.

4. Preliminaries

In this chapter, we are concerned with basic questions of the computational complexity of multiobjective optimization problems. In Section 4.1 we reflect a few thoughts on the theory of **NP**-hardness in multiobjective optimization and how some results seem to mislead the general picture of **NP**-hardness of multiobjective optimization problems. In Section 4.2, we introduce the theory of output-sensitive complexity as a formal framework to investigate algorithms and their performance. In Section 4.3 we discuss connections to another complexity measure for multiobjective optimization, namely Smoothed Analysis. Concluding this chapter, in Section 4.4, we show a first positive result that the MOMC problem is an easy problem in this framework by demonstrating a sufficient condition for MOCO problems being easy which can be derived from the existing literature.

4.1. Traditional NP-Hardness Theory

In multiobjective optimization the notion of **NP**-hardness has been adopted since the pioneering work by P. Serafini in 1986 [124]. Many papers cite Serafini to show that the multiobjective version of the shortest path, matching or matroid optimization problem are hard to solve. We take the multiobjective shortest path problem as an example how **NP**-hardness is usually applied in the multiobjective optimization literature.

One of the first papers which is concerned with the MOSP problem is the work by P. Hansen [74]. He defines a version of the biobjective shortest path problem, i.e., the MOSP with $d = 2$ and finds the first proof of intractability. He also mentions that intractability must not be confused with **NP**-hardness as it can still be possible to solve an **NP**-hard problem "in polynomial time in the very improbable case of $\mathbf{P} = \mathbf{NP}$" (cf. Hansen [74]).

A. Warburton [138] and P. Serafini [124] are the first authors who discuss a generalization of the canonical decision problem (cf. Definition 2.5) for multiobjective optimization problems:

Definition 4.1 (Canonical Decision Problem (MO Version))**.** For a multiobjective optimization problem $O = (I, S, v)$, the corresponding *canonical decision problem* O^{DEC} is the following set:

$$\{(x, \boldsymbol{k}) \in I \times \mathbb{Q}^d \mid \exists s \in S(x) : v(x, s) \leq \boldsymbol{k}\}$$

Both authors show that $MOSP^{\text{DEC}}$ is **NP**-hard and state this as an informal evidence that MOSP is a hard problem, while they also state that MOSP is actually intractable.

Later, in many papers, e.g., Galand et al. [67], Guerriero and Musmanno [71], Horoba [77], Mohamed, Bassem, and Taicir [99], Müller-Hannemann et al. [100], Raith and Ehrgott [115], Sanders and Mandow [120], Shekelyan, Jossé, and Schubert [125], Shekelyan et al. [126], Skriver [128], Tarapata [132], and Tsaggouris and Zaroliagis [134], it is claimed that MOSP itself is **NP**-hard. Therein, the arguments are based on

1. the paper by Serafini, or

2. the argument is given that since the output is exponential, the problem is **NP**-hard.

Concerning (2), this argument is not a formal proof of **NP**-hardness and R. Ladner already showed in 1975 [91] that if $\mathbf{P} \neq \mathbf{NP}$ then there exist infinitely many problems which are neither **NP**-hard nor polynomial-time solvable. In fact, it is what Hansen warned against, that **NP**-hardness should not be confused with intractability. It is by no means clear, why an output of exponential size implies that there is a polynomial-time reduction from every problem in **NP** to MOSP. Similar examples can be found for the multiobjective versions of the assignment, the spanning tree and matroid optimization problems.

Let us investigate the first argument and first recap how **NP**-hardness is usually derived in single objective optimization. We usually prove **NP**-hardness of an optimization problem by showing the **NP**-hardness of the canonical decision problem (cf. Definition 2.5). Let us consider the TSP as an example: To show that single-objective TSP is **NP**-hard, we show that TSP^{DEC} is **NP**-hard. Recall, that TSP^{DEC} is the decision problem whether there exists a feasible tour of cost less or equal to a given number $k \in \mathbb{Q}$. The classical reduction is by reduction from the Hamiltonian cycle problem: In the Hamiltonian cycle problem, we are given a (not necessarily complete) graph G and have to decide if there exists a Hamiltonian cycle in the graph. We construct an instance of the TSP problem by copying the graph and giving each edge the cost 1. Then, to make the graph complete, we add all remaining edges and give these edges cost 2. Thus, there exists a Hamiltonian cycle in G iff there exists a feasible tour of costs at most n in the new graph. But why does showing that TSP^{DEC} is **NP**-hard show that TSP is **NP**-hard?

The key is that there is a canonical Cook-reduction from P^{DEC} to P for every (single objective) optimization problem P. Given an instance of P^{DEC}, we can solve the optimization problem and check if k is at least the value of an optimal solution. If it is, then there exists a solution with cost at most k, i.e., each optimal solution. If it is not then since every solution has at least the optimal cost which is already higher than k, there does not exist a solution with cost less than k.

If we now turn to multiobjective optimization we have the problem that the nondominated set, i.e., the output of an oracle, may be of exponential size. Thus, the same reduction does not seem to generally hold in the context of multiobjective optimization. Let us now look into the paper by P. Serafini and describe how his arguments fit in this matter.

4.1.1. The Paper by P. Serafini

Serafini is the first author conducting a deep complexity analysis of multiobjective combinatorial optimization problems in 1986. Serafini [124] investigates the computational complexity of the general multiobjective combinatorial optimization problem

$$\begin{aligned} \text{(S1)}\ \min\ & f(\boldsymbol{x}) \\ \text{s.t. }\ & \boldsymbol{x} \in F \end{aligned}$$

for $f\colon \{0,1\}^n \to \mathbb{Z}^d$, $F \subseteq \{0,1\}^n$ and $n, d \in \mathbb{N}$. He leaves the actual solution concept open. Observe that if the goal is to find the nondominated set, this is exactly the definition of MOCO problems in Definition 2.5.

Serafini is generally interested in problems for which the set of solutions F is too large to be scanned exhaustively. He also points out that sizes of nondominated sets can be exponential in n and thus they cannot be enumerated in polynomial time. So, he restricts the problems he studies to problems with nondominated sets of polynomial size by requiring that the maximum difference in solution values is bounded by a polynomial in the input, i.e.,

$$\max_{\boldsymbol{x},\boldsymbol{y}\in F} \|f(\boldsymbol{x}) - f(\boldsymbol{y})\|_\infty = K \in \mathcal{O}(n^k), \text{for some } k > 1. \tag{4.1}$$

He then shows reductions among several solution concepts of problem S1, e.g., finding the nondominated set only, finding all Pareto-optimal solutions and solving S1^{DEC}. He also proves that, assuming (4.1), finding the nondominated set of problem S1 is polynomial-time equivalent to S1^{DEC}.

He then puts forward three reasons why we can identify the complexity of O^{DEC} with the complexity of O in general:

1. They are equivalent if (4.1) holds.

2. The canonical decision problem is also studied in single objective optimization.

3. The canonical decision problem often plays a role in methods to solve MOCO problems in a posteriori or interactive approaches.

In the following sections, Serafini shows for several well-known MOCO problems that their canonical decision problem is **NP**-hard and, in the light of the above argumentation, states that the MOCO problem itself is **NP**-hard.

He starts by studying S1 with linear objective functions, which are linear MOCO problems by our definition, and for which the feasible set F contains the extreme points of a polyhedron P which again contains F, i.e., $\operatorname{vert} P \subseteq F \subseteq P$:

$$\begin{aligned} \text{(S2)}\ \min\ & C\boldsymbol{x} \\ \text{s.t. }\ & \boldsymbol{x} \in F \end{aligned}$$

for $C \in \mathbb{Z}^{d \times n}$. He investigates the complexity status of this general problem by looking at the BUCO problem (cf. Definition 2.15) as a special case of S2.

$$\begin{aligned} \text{(BUCO)} \quad & \min\ (\boldsymbol{c}^1, \boldsymbol{c}^2)^T \boldsymbol{x} \\ & \boldsymbol{x} \in \{0,1\}^n \end{aligned}$$

Serafini concludes that S2 is **NP**-hard, because BUCO^{DEC} is the binary knapsack problem, which is well-known to be **NP**-hard. He then goes on and investigates the biobjective shortest path and biobjective assignment problems as special cases of S2.

In the last section, Serafini proves **NP**-hardness for the biobjective matroid optimization problem and discusses neighborhoods induced by topological sorting and locally nondominated solutions.

4.1.2. Discussion of NP-hardness in multiobjective optimization

Let us first state that Serafini did write that S2, the biobjective shortest path, assignment and matroid optimization problems are **NP**-hard, but he proves it only for the respective canonical decision problems. The reason for this is the identification of a MOCO problem with its canonical decision problem, which was based on the informal reasons stated above and the equivalence of the problems under (4.1).

Hence, let us revisit the three reasons Serafini offers why a MOCO problem P can be identified with its canonical decision problem P^{DEC}. The third reason being that the hardness of P^{DEC} influences the choice of solution method to use to solve a problem at hand. This is a very important reason why the complexity status of P^{DEC} is interesting. But showing that a certain solution method is incapable of solving a problem efficiently does not determine the complexity status of the problem itself.

The second reason is that canonical decision problems are also investigated in single-objective optimization. But in single-objective optimization this is well justified as we discussed in the introduction to this section. And the justification might not transfer to the multiobjective case.

The first reason that the solution concept of finding the nondominated set and solving the canonical decision problem is equivalent under the bound (4.1) is solid on the first glance: If P^{DEC} is **NP**-hard under (4.1) then P is an **NP**-hard multiobjective optimization problem. But in his proofs of **NP**-hardness of the decision problems in the paper, Serafini does not assume (4.1) anymore. Since it is hard to prove that a problem is not **NP**-hard (proving that a problem in **P** is not **NP**-hard would imply $\mathbf{P} \neq \mathbf{NP}$), we give an example on which the above argumentation by Serafini fails.

Let us denote the special case of the BUCO problem with nonpositive c^1 and nonnegative c^2 by $BUCO_K$. The canonical decision problem ${BUCO_K}^{\text{DEC}}$ is the 0/1-knapsack problem and thus **NP**-hard in general without assuming (4.1). We already know that from this we cannot—in contrast to single objective optimization—deduce **NP**-hardness of $BUCO_K$ directly. But is there a way to deduce **NP**-hardness of ${BUCO_K}^{\text{DEC}}$ under (4.1)? We show that this is not the case by proving that $BUCO_K$ is not **NP**-hard under (4.1) unless $\mathbf{P} = \mathbf{NP}$. This is a strong evidence that the

assumption that from **NP**-hardness of P^{DEC} follows **NP**-hardness of P under (4.1) for MOCO problems P does not hold. We prove this by showing that the Nemhauser-Ullmann algorithm (cf. Nemhauser and Ullmann [102]) for the knapsack problem solves BUCO in polynomial time if (4.1) is assumed.

Let us first discuss the Nemhauser-Ullmann algorithm in some detail. We are given an instance of the knapsack problem, i.e., $\boldsymbol{c}^1, \boldsymbol{c}^2 \in \mathbb{N}^n, W^1, W^2 \in \mathbb{N}$ and we are to decide whether there exists $\boldsymbol{x} \in \{0,1\}^n$ with ${\boldsymbol{c}^1}^T\boldsymbol{x} \geq W_1$ and ${\boldsymbol{c}^2}^T\boldsymbol{x} \leq W_2$. We use the following interpretation: The vector $(\boldsymbol{c}_i^1, \boldsymbol{c}_i^2)^T$ encodes the gain and loss if we pack item $i \in [n]$, thus we can view this decision problem as a biobjective optimization problem with objective functions $\min -{\boldsymbol{c}^1}^T\boldsymbol{x}$ and $\min {\boldsymbol{c}^2}^T\boldsymbol{x}$ and get a $BUCO_K$-instance. We observe that if there is a feasible packing, then there is one which is also Pareto-optimal.

The algorithm follows a dynamic programming scheme: Assuming an arbitrary ordering of the items, we look at the Pareto-optimal packings for the first i items. We get the Pareto-optimal packings for the first $i+1$ items by adding the $(i+1)$-th item to each of our packings, retaining the old ones. We call a packing $\boldsymbol{x} \in \{0,1\}^n$ *dominated* if there is another packing $\boldsymbol{x}' \in \{0,1\}^n$ with ${\boldsymbol{c}^1}^T\boldsymbol{x}' \geq {\boldsymbol{c}^1}^T\boldsymbol{x}$, ${\boldsymbol{c}^2}^T\boldsymbol{x}' \leq {\boldsymbol{c}^2}^T\boldsymbol{x}$, ${\boldsymbol{c}^1}^T\boldsymbol{x}' \neq {\boldsymbol{c}^1}^T\boldsymbol{x}$, and ${\boldsymbol{c}^2}^T\boldsymbol{x}' \neq {\boldsymbol{c}^2}^T\boldsymbol{x}$. We can now delete all packings which are dominated by another packing we already produced, since they are never a subsolution of a Pareto-optimal packing. To solve the knapsack problem, we can delete all solutions for which ${\boldsymbol{c}^2}^T x > W_2$. This algorithm is well known to be pseudo-polynomial and has a running time of $\mathcal{O}(nW_2)$.

Proposition 4.1. *If $BUCO_K$ is **NP**-hard assuming (4.1) then **P** = **NP**.*

Proof. To solve the $BUCO_K$ problem, we set $W_2 \coloneqq {\boldsymbol{c}^2}^T\mathbf{1}$ and thus keep all Pareto-optimal solutions. Observe, that packing nothing, i.e., the solution $\boldsymbol{x} = \mathbf{0}$, is always a Pareto-optimal packing. Using this and because of (4.1), we have for some $k > 1$

$$\begin{aligned}
\mathcal{O}(n^k) \ni K &= \max_{\boldsymbol{y},\boldsymbol{z} \in F} \|f(\boldsymbol{y}) - f(\boldsymbol{z})\|_\infty \\
&= \max_{\boldsymbol{x} \in F} \|C\mathbf{0} - C\boldsymbol{x}\|_\infty \\
&= \max_{\boldsymbol{x} \in F} \|C\boldsymbol{x}\|_\infty \\
&= \max\{\sum_{i \in [n]} \boldsymbol{c}_i^1, \sum_{i \in [n]} \boldsymbol{c}_i^2\} \geq \boldsymbol{c}_i^j, \text{ for } j \in \{1,2\} \text{ and } i \in [n].
\end{aligned}$$

And since $nW_2 = n \cdot {\boldsymbol{c}^2}^T\mathbf{1} \in \mathcal{O}(n^{k+1})$, the Nemhauser-Ullmann algorithm runs in polynomial time on $BUCO_K$ instances. □

As for the other problems, the MOSP problem, the multiobjective assignment and biobjective matroid optimization problem, it is not obvious what happens if (4.1) is assumed. Regarding the MOSP problem, there do exist pseudo polynomial time algorithms based on the generation of Pareto-optimal paths from s to the other nodes (cf. e.g., Martins [98]). Using the bound (4.1), we can restrict the size of the labels

at the target node t, but we are not able to directly bound the number of candidate paths at intermediate nodes $v \in V \backslash \{s, t\}$. Moreover, the bound we showed in the proof of Proposition 4.1 does not apply. Later in this thesis we show a new proof of **NP**-hardness of the MOSP problem in Section 6.1.

In conclusion, while the **NP**-hardness of a canonical decision problem is interesting, it does not seem to have (mathematical) implications on the **NP**-hardness of the corresponding MOCO problem. Further, we might even want to ask the question what we can draw from the insight of a MOCO problem with exponential nondominated set actually being **NP**-hard. On one hand, **NP**-hardness of a problem is a good reason to believe that the problem cannot be solved in polynomial time. On the other hand, intractability proves that there is no polynomial-time algorithm in general.

The theory of **NP**-hardness in multiobjective optimization is still very interesting if we cannot rule out polynomial-time algorithms by other means. The bound by Serafini is one possibility to restrict problems to nondominated sets of polynomial size. A more theoretical option is to restrict a problem to all instances which have polynomial nondominated sets. But similar to Serafini's bound (4.1), it might be impossible to test this property in polynomial time in general and thus even testing the feasibility of an instance becomes hard.

It could also be interesting to modify MOCO problems by imposing an ordering on the nondominated set to output. That is, requiring to output the nondominated set of a MOCO problem in an ordering which is part of the problem, i.e., output the nondominated set in lexicographic order. Then again, **NP**-hardness can be an interesting tool even though the output is exponential in the worst-case.

In this thesis, we suggest another framework of assessing computational complexity of multiobjective optimization problems, i.e., output-sensitive complexity.

4.2. Output-Sensitive Complexity Theory

In this section, we give basic definitions of output-sensitive complexity also called enumeration complexity. The main difference to classic complexity theory is that the running time is not only measured in the input size but also in the size of the output. The formal terms were used for a long time now, a first summary can be found in the work by Johnson, Yannakakis, and Papadimitriou [79]. In 2009 in a Master's thesis, J. Schmidt [121] gives a complete formal framework of enumeration complexity which we reproduce here.

The main point is that we can view the MOCO problems defined in Section 2.5 as enumeration problems. We also prove this formally in Subsection 4.2.2 to make the relationship between MOCO problems and enumeration problems mathematically sound. And we give a new definition of tractability of multiobjective optimization problems.

4.2.1. Formal Definitions

In this section, we give formal definitions of the terms enumeration problem and enumeration algorithm. Moreover, we define complexity classes for enumeration problems. This section is largely based on the Master's thesis by Schmidt [121] and was published in a paper by Bökler et al. [18].

Definition 4.2 (Enumeration Problem, cf. Schmidt [121, pp. 7 sq.])**.** An *enumeration problem* is a pair (I, C) such that

1. $I \subseteq \Sigma^*$ is the set of *instances* for some fixed alphabet Σ,
2. $C\colon I \to 2^{\Sigma^*}$ maps each instance $x \in I$ to its *configurations* $C(x)$, and
3. the encoding length $|s|$ for s in $C(x)$ for x in I is in $\text{poly}(|x|)$.

We assume that I is decidable in polynomial time and that C is computable.

We usually assume that the alphabet Σ is $\{0, 1\}$. This is important, because all instances and all configurations need to be encoded by finite strings, while the exact encoding is not relevant here. The third requirement means, that the configurations need to be compact, i.e., the encoding length of a configuration of an instance x should be polynomially bounded in the size of x. The reason for this is to limit the number of configurations to be at most exponentially many and to not make the problems artificially hard by allowing the output of one configuration to be a hard problem.

Definition 4.3 (Enumeration Algorithm, cf. Schmidt [121, p. 11])**.** Let $E = (I, C)$ be an enumeration problem. An *enumeration algorithm* $\mathcal{A}$ for E is a RAM that

1. on input x in I outputs each c in $C(x)$ *exactly once*, and
2. on every input terminates after a *finite* number of steps.

Let $x \in I$ and let $|C(x)| = k$, then the 1st *delay* is the number of steps of $\mathcal{A}$ before the first configuration is output, the ith delay for $i \in \{2, \ldots, k-1\}$ is the number of steps of $\mathcal{A}$ between the output of the $(i-1)$th and the i-th configuration, and the $(k+1)$th delay is the number of steps of $\mathcal{A}$ between the last output and the termination of the algorithm.

The following complexity classes can be used to classify enumeration problems, cf. also [79].

Definition 4.4 (Output-Sensitive Complexity Classes, cf. Schmidt [121, p. 12])**.** Let $E = (I, C)$ be an enumeration problem. Then E is in

1. **TotalP** (Output Polynomial Time/Polynomial Total Time)[1],
2. **IncP** (Incremental Polynomial Time),

[1] Although we use the term *output polynomial time* in the remaining text, we abbreviate it to **TotalP** for historical and notational reasons.

3. **DelayP** (Polynomial Time Delay),
4. **PSDelayP** (Polynomial Time Delay with Polynomial Space)

if there is an enumeration algorithm such that

1. its running time is in $\text{poly}(|x|, |C(x)|)$ for x in I,
2. its i-th delay for i in $[|C(x)| + 1]$ is in $\text{poly}(|x|, i)$ for x in I,
3. its i-th delay for i in $[|C(x)| + 1]$ is in $\text{poly}(|x|)$ for x in I, and
4. the same as in 3. holds and the algorithm requires at most polynomial space in the input size,

respectively.

We say that an algorithm is *output-sensitive* if it fulfills condition 1 of the above definition. Moreover, we say that an algorithm has *incremental delay* (*polynomial delay*) if it fulfills the second (third) condition of the above definition. An enumeration algorithm is regarded *efficient* if it is output-sensitive. As an overview on these classes, the following hierarchy result holds.

Theorem 4.2 (Schmidt [121, pp. 14 sqq.])**.**

$$\boldsymbol{PSDelayP} \subseteq \boldsymbol{DelayP} \subseteq \boldsymbol{IncP} \subset \boldsymbol{TotalP}.$$

4.2.2. Connection to Multiobjective Optimization

We now explain, how MOCO problems as defined in Definition 2.5 relate to enumeration problems as defined in Definition 4.2. More formally, we show that the general MOCO problem is an enumeration problem. Given a MOCO instance $O = (I^O, S^O, v^O)$ we need to define the set I of instances of the enumeration problem and the configuration mapping C.

To define I, we identify it with I^O. The configuration mapping now maps an instance to its nondominated set $\mathcal{Y}_N$. Let us fix an arbitrary encoding of rational numbers with at most $\mathcal{O}(\langle r \rangle)$ bits for a rational number r. We observe that the value of each component j of each point of the nondominated set is at most $\sum_{i=1}^{n} |C_{ji}| \in \mathcal{O}(\langle C \rangle)$ and thus the encoding of each point in the nondominated set is polynomial in the instance encoding size. Thus, the configurations of a MOCO problem are the points of the nondominated set and not, for example, the solutions of the MOCO problem.

Solving the so defined enumeration problem gives us the set of configurations, i.e., the nondominated set in the given encoding. This corresponds exactly to our definition of solving a MOCO problem.

Further, we can now define what an efficient algorithm for a MOCO problem in this framework is: We say that an algorithm for a multiobjective optimization problem is efficient if it is output-sensitive for a fixed number of objectives. Fixing the number of objectives is a reasonable assumption, since we usually deal with a small number in practice and are interested in the behavior of solution methods for this specific number of objectives.

4.3. Connection to Smoothed Analysis for MOCO Problems

Before we move forward to using output-sensitive complexity, we need to address one practical objection regarding MOCO problems. It is often argued that computing the entire nondominated set of a MOCO problem is too costly to pursue, because it can have an exponential size in the input size. But in practice, it is observed that usually the situation is not so bad when the number of objectives is small. This observed discrepancy between practice and the traditional worst-case analysis motivates stochastic running time analyses, where the inputs are drawn from a certain distribution. One prominent stochastic running time analysis framework, so called *Smoothed Analysis*, can be used to bound the expected worst-case size of the nondominated set of a MOCO problem.

In classical running time analysis, we play against an adversary who gives us ill posed instances in the sense that the running time is very high or the nondominated set is very large. Smoothed Analysis aims at weakening this adversary. In the context of multiobjective optimization, a model by Ackermann et al. [1] was used to show the best expected bound on the smoothed worst-case size of the nondominated set of general MOCO problems.

Let (I, S, v) be a MOCO problem. In worst-case running time analysis, the adversary is allowed to choose an instance $x \in I$ arbitrarily. In the model by Ackermann et al. [1], the adversary is only allowed to choose the first row of it deterministically and for all remaining rows $i \in \{2, \dots, d\}$ and columns $j \in [n]$ it may choose a probability density function $f_{i,j} : [0, 1] \to \mathbb{R}$, describing how potential entries are drawn. The adversary is not allowed to choose these densities arbitrarily, because otherwise it could again choose the coefficients deterministically. Rather, the $f_{i,j}$ are bounded by a model parameter $\phi \geq 1$. The adversary chooses the set of solutions $S \subseteq \{0, 1\}^n$ completely arbitrarily. Brunsch and Röglin [26] showed that the expected size of the nondominated set of these instances is at most $\mathcal{O}(n^{2d}\phi^d)$.

Thus, when we fix the number of objectives, we can expect to have only polynomial nondominated set sizes of any MOCO problem in practice. This emphasizes the need for output-sensitive algorithms, because when the nondominated set is small, we want our algorithms to be fast and when the nondominated set is large, we want them to be not too slow.

We can also investigate the connection between smoothed complexity and output-sensitive complexity of MOCO problems. At first we can observe that when we have a **TotalP**-algorithm with a running-time polynomial in the input and the size of the nondominated set, then by the above observations the algorithm is also a smoothed-polynomial algorithm for every fixed number of objectives.

Corollary 4.3. *If a MOCO problem O is in **TotalP**, then it can be solved in smoothed polynomial time for every fixed number of objectives.*

The converse is most probably not true: In Section 6.1 we see that if $\mathbf{P} \neq \mathbf{NP}$ then there is no output-sensitive algorithm for the MOSP problem even in the case of 2

objectives. But it is easy to show that, e.g., Martins' algorithm (cf. Martins [98]) has indeed a smoothed polynomial running-time.

Corollary 4.4. *Not every MOCO problem that can be solved in smoothed polynomial time can also be solved in output-polynomial time, unless* $\boldsymbol{P} = \boldsymbol{NP}$.

4.4. A Positive Example

In this section, we show a sufficient condition for a MOCO problem being solvable in output polynomial time, which follows directly from the multiobjective optimization literature. A broad subject of investigation in the multiobjective optimization literature is the $\boldsymbol{\varepsilon}$-constraint scalarization. Given a MOO problem it is the following single objective optimization problem:

Definition 4.5 ($\boldsymbol{\varepsilon}$-Constraint Scalarization)**.** The ε-constraint scalarization of a MOO (I^M, S^M, v^M) is the following single objective optimization problem (I, S, v): The instances are the set $I = I^M \times \mathbb{Q}^d$, the solution mapping is the function

$$S((x, \boldsymbol{\varepsilon})) = \{s \in S^M(x) \mid v^M(x, s) \leq \boldsymbol{\varepsilon}\}$$

and the value function is $v((x, \boldsymbol{\varepsilon}), s) = \mathbf{1}^T v^M(x, s)$.

In other words, we minimize the sum of the objectives and restrict the feasible set to those solutions which have cost less or equal to $\boldsymbol{\varepsilon}$ (component wise). It is well known that the complete nondominated set can be found using this scalarization, cf. [29], but the scalarization itself can be hard to solve, cf. [56]. In the past, authors have proven several bounds on the number of $\boldsymbol{\varepsilon}$-constraint scalarizations needed to find the nondominated set of a general MOCO problem. It is possible to construct a sufficient condition for enumerating the nondominated set of a MOCO problem in output polynomial time from these results.

One of the first such bounds is due to Laumanns, Thiele, and Zitzler [92], who prove that at most $\mathcal{O}(|\mathcal{Y}_N|^{d-1})$ single-objective $\boldsymbol{\varepsilon}$-constraint scalarization instances need to be solved. The currently best bound is due to Klamroth, Lacour, and Vanderpooten [87], who prove a bound of $\mathcal{O}(|\mathcal{Y}_N|^{\lfloor \frac{d}{2} \rfloor})$.

From these results we can formulate the following corollary.

Corollary 4.5. *If the* $\boldsymbol{\varepsilon}$*-constraint scalarization of a given MOCO problem* P *can be solved in polynomial time for a fixed number of objectives,* $\mathcal{Y}_N$ *can be enumerated in output-polynomial time for a fixed number of objectives.*

One example for such a problem is the MOMC problem. Armon and Zwick [7] showed that the $\boldsymbol{\varepsilon}$-constraint version of the problem can be solved in time $\mathcal{O}(|E||V|^{2d} \log |V|)$. Thus, the MOMC is an example of a multiobjective optimization problem that can be solved in output polynomial time for each fixed number of objectives.

Now, the question arises if this condition is also necessary, i.e., can we show that a MOCO problem cannot be solved in output polynomial time by showing that the

ε-constraint scalarization cannot be solved in polynomial time. But as the following proposition shows, if $\mathbf{P} \neq \mathbf{NP}$ the condition of Corollary 4.5 is not necessary.

Proposition 4.6. *There is a MOCO problem $P \in$ **IncP** with P^{DEC} being **NP**-complete.*

Proof. We consider the following biobjective problem

$$(P)\ \min\{(\boldsymbol{c}^T\boldsymbol{x}, -\boldsymbol{c}^T\boldsymbol{x}) \mid \boldsymbol{x} \in \{0,1\}^n\}$$

with $\boldsymbol{c} \in \mathbb{N}^n$. We first show **NP**-hardness and membership in **NP** of P^{DEC} and then show membership in **IncP** of P.

Recall that P^{DEC} is defined as the decision problem in which we are given an instance x of $P = (I, S, v)$ and a vector $k \in \mathbb{Q}^d$ and the goal is to decide whether there exists an $s \in S(x)$ such that $v(x, s) \leq k$. Setting

$$k \coloneqq \frac{1}{2}\mathbf{1}^T c \begin{pmatrix} 1 \\ -1 \end{pmatrix},$$

we can reformulate P^{DEC} in the following way: Decide for a given vector $\boldsymbol{c} \in \mathbb{N}^n$ if there is an $\boldsymbol{x} \in \{0,1\}^n$ such that $\boldsymbol{c}^T\boldsymbol{x} = \frac{1}{2}\sum_{i=1}^n \boldsymbol{c}_i$. We observe that this decision problem is the well-known partition problem shown by Garey and Johnson [69] to be **NP**-hard. It follows from Definition 4.1 that P^{DEC} is in **NP** and thus P^{DEC} is **NP**-complete.

Enumerating the nondominated set of P can be done in incremental polynomial time by employing a recursive enumeration scheme on the first i variables (fixing an arbitrary variable-ordering). Setting all variables to 0 yields point $(0,0)^T$. Now we fix all variables after the i-th to 0 and allow the first i variables to vary. Let F_i be the nondominated set for this restricted problem. If we know F_i, we can compute F_{i+1} by computing $F_i \cup (F_i + \{\boldsymbol{c}_{i+1}\})$. Each such step yields time $\mathcal{O}(|F_i| \log |F_i|)$ and we have $F_0 \subsetneq F_1 \subsetneq \cdots \subsetneq F_n$, i.e., we find at least one new point in each iteration. □

This concludes the first chapter. We now look into other positive results in the field of output-sensitive complexity of multiobjective optimization problems.

5. Multiobjective Linear Optimization and Nondominated Extreme Points

The motivation for this chapter is the following observation: In biobjective problems, there exist very efficient practical algorithms to compute $\mathcal{Y}_X$, i.e., the extreme points of the nondominated set. While experience often shows that the complete nondominated set $\mathcal{Y}_N$ can be hard to compute. This observation gave rise to a solution methodology called the *two-phases method* (cf. e.g. Ulungu and Teghem [136]), where a fast method to find the extreme points of the nondominated set is applied and then a search for the remaining nondominated points is conducted. But the two-phases method remained only usable in the biobjective case, because of the lack of algorithms to compute the nondominated extreme points of problems with more than two objectives. M. Ehrgott and X. Gandibleux address the need for practical algorithms to find nondominated extreme points in their survey paper from 2000 [57] as a "first step to an application of the two phases method in three or more criteria MOCO".

There is a methodology which, at least in theory, is able to compute the extreme points of the nondominated set of a MOCO problem by using the convex relaxation and algorithms for MOLP. Let us define what a *convex relaxation* is:

Definition 5.1 (Convex Relaxation)**.** Given a MOCO problem (I, S, v), an instance $x \in I$ with objective function matrix C_x, the convex relaxation is the following MOLP:

$$\begin{aligned} \min\ & C_x \boldsymbol{x} \\ & \boldsymbol{x} \in \operatorname{conv} S(x) \end{aligned}$$

Now we can observe that the problem MOLP^{YEx} is exactly the problem of finding the nondominated extreme points of the given MOCO problem. The concept of a convex relaxation was introduced by Cerqueus, Przybylski, and Gandibleux [28] and Ehrgott and Gandibleux [58]. Note though, that this MOLP is not in the standard form of Definition 2.21, since we need to be able to describe the feasible polyhedron by means of linear inequalities. But if we were able to describe the integer hull of every binary linear programming problem by linear inequalities of polynomial size in polynomial time, then essentially $\mathbf{P} = \mathbf{NP}$. Moreover, even problems in $\mathbf{P}$ might have only complex integer hulls as T. Rothvoss proved [117]. So this method is usable if we have a compact MOLP formulation for our given MOCO problem.

Previous Work on MOLP and Finding Nondominated Extreme Points

Most approaches to the MOLP problem are based on pivoting of basic feasible solutions in the style of the traditional simplex algorithm by G. Dantzig (cf., e.g., [39]). Examples

for such approaches are by Armand [5], Armand and Malivert [6], Ecker, Hegner, and Kouada [51], Ecker and Kouada [52], Evans and Steuer [62], Gal [66], Isermann [78], Philip [112, 113], Rudloff, Ulus, and Vanderbei [118], Schönfeld [122], Strijbosch, Doorne, and Selen [131], Yu and Zeleny [139, 140], and Zeleny [141]. For an in-depth introduction to these algorithms, we refer the reader to the book by Ehrgott [55]. All these approaches have in common that running time analyses are not considered.

Another approach is based on the weights for weighted-sum linear programming problems $P_1(\ell)$. For each extreme point $\boldsymbol{y}$ of the upper image, there is a set of weights Λ, such that for every $\ell \in \Lambda$ the LP $P_1(\ell)$ has an optimal solution $\boldsymbol{x}$ with $C\boldsymbol{x} = \boldsymbol{y}$. A central theory around these weight spaces is formulated by Benson and Sun [14].

In recent years, new objective space centered methods evolved. Examples for these approaches are Dauer [40], Dauer and Gallagher [41], Dauer and Liu [42], and Dauer and Saleh [43]. And also the work by H. Benson [11–13] and since approaches spawned by his work are the major concern in this chapter, they are discussed in detail in Section 5.1.

Today, also implementations are available to support practitioners in solving MOLP problems. One of the first available is from A. Löhne and B. Weißing (cf. [97] and http://www.bensolve.org) which is an implementation of the Primal and Dual Benson algorithm as described later. Another implementation from the Zuse Institute Berlin is also able to solve multiobjective mixed integer programs and the core MOLP solver is similar on the Dual Benson algorithm (cf. http://polyscip.zib.de and Borndörfer et al. [21]). A very recent project is vOptSolver by X. Gandibleux and S. Ruzika which aims at providing a solver suite for multiobjective optimization problems. Implementations for MOLP and MOCO will be available over time (cf. https://vopt-anr-dfg.univ-nantes.fr).

Regarding the problem of finding nondominated extreme points of MOCO problems (or the more general case of bounded multiobjective integer problems), there exist efficient methods in theory and practice for the case of two objectives. The method, today usually called the *dichotomic approach*, was first introduced independently by Cohen in 1978 [34], Aneja and Nair in 1979 [4], and Dial also in 1979 [48]. Especially, Aneja and Nair showed that if we have access to an algorithm $\mathcal{A}$ which solves a lexicographic version of the MOCO problem, we can find the set of nondominated extreme points of size $k \geq 2$ by needing $2k - 1$ calls to $\mathcal{A}$.

In the case of three and more objectives there are only two approaches known: The approaches by Özpeynirci and Köksalan [106] and Przybylski, Gandibleux, and Ehrgott [114]. Özpeynirci and Köksalan propose an algorithm which is inspired by the dichotomic approach. The algorithm maintains stages which are similar to the pairs of extreme points in the dichotomic approach—but in dimension d, a stage consists of d points. The stages are then tested for new extreme points similarly to the dichotomic approach. But in contrast to the dichotomic approach, some of these stages can yield negative and thus invalid weights. New stages are constructed from old stages and potential new extreme points. Özpeynirci and Köksalan do not conduct a running-time analysis, but provide some experiments for the case of three and four objectives. This algorithm is a competitor to our implementation in the computational

study in the end of this chapter.

The other approach by Przybylski, Gandibleux, and Ehrgott works on the weight-space decomposition of the given problem instance. It decomposes the set of weights $W = \{\boldsymbol{\ell} \in \mathbb{R}^{d-1}_{\geq} \mid ||\boldsymbol{\ell}||_1 \leq 1\}$ into polytopes, where each polytope is the set of weights such that each such weight yields the same extreme point of the nondominated set when used in a weighted-sum LP. The decomposition is then constructed in a geometric manner by inspecting the facets of these polytopes. In the special case of three objectives, Przybylski, Gandibleux, and Ehrgott show an efficient way how this can be executed. For more than three objectives, they show how this case can be recursively reduced to a lower dimensional case. While this procedure is very efficient in the case of three objectives as is also demonstrated in computational experiments in the paper, the authors state that even the implementation for the general case is too complicated to pursue.

Contributions and Organization

We first introduce the algorithm and its background theory in Section 5.1. In Section 5.2, we investigate the original or primal algorithm and its running time. We see that the original algorithm suffers from two problems: We cannot control the encoding length of the numbers in the input easily and many redundant faces can potentially be enumerated. To overcome these problems, we introduce an improvement of the algorithm which allows us to reduce the running time significantly and also to control the encoding length of numbers in the input. The result is that we can enumerate facets of the upper image very efficiently and also bound the delay of the algorithm, while still giving a decent running time for enumerating the extreme points of the upper image.

In Section 5.3, we are concerned with the so called Dual Benson algorithm. We conduct a running time analysis based on a slightly modified version of the algorithm as presented in the literature. Again, we see that the algorithm suffers from enumerating redundant objects and we suggest an improvement that overcomes these problems. The result is symmetric to the result of the primal algorithm: We can bound the delay in enumerating the extreme points of the upper image and still get a good overall running time for enumerating the facets of the upper image.

Further, we show that by using this algorithm, we can find nondominated extreme points for a very wide range of MOCO problems without detouring round multiobjective linear programming in Section 5.4. We also demonstrate the practical impact of our implementation of the algorithm and compare it to the one existing state-of-the-art implementation for an arbitrary number of objectives. Subsequently, we see that our Dual Benson implementation is capable of solving moderately sized instances with up to six objectives which was not possible before.

5.1. The Benson Algorithm Family

The algorithms of the Benson algorithm family are descendants of an algorithm which was introduced by H. Benson in 1998 [11]. Benson notes in his paper that there is a need for algorithms working in the output space as opposed to most of the previous approaches which almost exclusively worked on the decision space. A first example of an implemented algorithm for MOLP was the implementation of a multiobjective simplex method by R. Steuer [130] which was called ADBASE from 1975. It was the first time researchers could solve MOLP instances and at the same time it was a prime example of its limitations: Benson writes in his paper that even a problem with 60 variables and 50 linear inequality constraints could not be handled by the implementation on the given hardware from 1998. One reason was that the number of extreme points in decision space it needed to visit was very large. Hence, a driving force to look into objective space methods was the observation that the number of nondominated extreme points in the objective space is at most the number of Pareto-optimal extreme points in the decision space.

The algorithm H. Benson describes maintains an outer approximation polyhedron of the image of the feasible polyhedron under the objective function. In each iteration an inequality is added which cuts this outer approximate polyhedron, still ensuring this new polyhedron to be an outer approximation. New inequalities can be found by solving linear programming problems. A computationally very demanding procedure needed in the algorithm is to switch between inequality and vertex representations of the outer approximate polyhedron. This problem is well known as the vertex enumeration problem (cf., e.g., Avis and Fukuda [9], Bremner [22], Bremner [23], Bremner, Fukuda, and Marzetta [24], Chazelle [30], Dyer [50], and Fukuda and Prodon [65]) and is a hard enumeration problem in general (cf. Khachiyan et al. [85]). In this way, the algorithm computes a description consisting of extreme points and inequalities of the nondominated subset of the image of the input polyhedron under the objective function.

The algorithm as described by H. Benson does have an assumption on the instances: The ideal point needs to exist for the given input instance, otherwise the initial outer approximation cannot be constructed trivially. While this excludes certain MOLP instances, this restriction is not very limiting in practice. For example, if the MOLP describes the integer hull of a MOCO problem, an ideal point exists iff the instance has a feasible solution. Also for an instance of a multiobjective integer problem, the ideal point always exists if the nondominated set itself is nonemtpy and finite.

In the last 20 years after its invention, the algorithm was tested in applications in finance and also in information theory (cf. Csirmaz [38]). Since the algorithm got more and more popular, also more improvements were proposed regarding the number of linear programs that need to be solved, see for example Csirmaz [38] and Hamel, Löhne, and Rudloff [73]. Another point of interest is the way the output space is handled (Burton and Özlen [27]). And also H. Benson himself wrote a paper about the theoretical properties of his algorithm, cf. Benson [12].

A Dual Algorithm

For single-objective linear programming and its applications in combinatorial optimization, linear programming duality is a crucial tool. A similar theory for MOLP which is easy to apply and with a geometrical interpretation was lacking until the work by F. Heyde und A. Löhne [75] in 2008. Hyde and Löhne introduce a geometrically dual polyhedron which is closely related to duality theory in single-objective linear optimization. Based on the upper image in MOLP (which was also defined by Heyde and Löhne), they define the geometrically dual polyhedron and baptize it the *lower image*. A very curios detail of this duality are its geometric properties: An extreme point of the upper image can be bijectively mapped to facets of the lower image and, in general for an upper image of dimension d, every i-dimensional face of the upper image can be bijectively mapped to a $((d-1)-i)$-dimensional face of the lower image for $i \in [d-1]$. Moreover, this mapping is easy to compute. This was possible by generalizing MOLP to vector optimization problems; so in a narrow sense, the dual problem is not a MOLP, but a vector optimization problem. We do not go into the details of vector optimization and general ordering-cones in this thesis, but we note that the work by Heyde and Löhne was necessary to formulate the dual algorithm.

The dual algorithm was proposed in 2012 by M. Ehrgott, A. Löhne and L. Shao [59]. Instead of computing a description of the upper image, the dual algorithm computes an extreme point and inequality representation of the lower image. From this perspective the dual algorithm works in the same way as the original or primal algorithm: An outer approximation of the lower image is maintained and refined by means of new cutting inequalities. As in the primal algorithm, vertex enumeration is a fundamental building block. Due to the geometric duality, finding an inequality and extreme point representation of the lower image gives us an inequality and extreme point representation of the upper image.

The algorithm is also of interest from the perspective of weight-set decompositions (cf., e.g., Benson and Sun [14]). The dual polyhedron can be interpreted as a lifting of the weight-set decomposition into a space with one additional component. The additional component describes the optimal value of the weighted-sum linear program with the weights given by the other components. This observation led to the implementation used in PolySCIP (cf. Borndörfer et al. [21]).

Another interpretation of the dual algorithm is an inner approximation of the upper image: The dual of the intermediate polyhedra maintained in the dual algorithm are inner approximate polyhedra of the upper image. In each iteration we thus find a new face of the upper image and solve a problem which is dual to the vertex enumeration problem, sometimes called the facet enumeration or convex hull problem.

We now go into more details of both of the primal and dual algorithm, perform running time analyses on both and also propose improvements to both algorithms.

5.2. Finding Facets: Benson's Algorithm

We now follow Ehrgott, Löhne, and Shao [59] and Hamel, Löhne, and Rudloff [73] in describing the primal algorithm in more detail. A pseudocode listing can be found in Listing 1.

So let an MOLP with constraint matrix $A \in \mathbb{Q}^{m\times n}$, an objective function matrix $C \in \mathbb{Q}^{d\times n}$ and a left hand side vector $\boldsymbol{b} \in \mathbb{Q}^m$ be given. In an initialization phase (cf. Lines 2 to 5), the first step is to compute an initial polyhedron which contains the upper image (cf. Section 2.5.1) $\mathcal{P}$ entirely. This polyhedron is $L = y_{\min} + \mathbb{R}^d_{\geq}$, where $y_{\min}$ is the ideal point; this polyhedron can be represented by the d inequalities $\boldsymbol{y}_i \geq (y_{\min})_i$ for $i \in [d]$. Accordingly, the algorithm computes $y_{\min}$ by solving the weighted-sum linear programs $P_1(\boldsymbol{e}_i)$, for $i \in [d]$ and stores the initial inequalities. If one of the programs $P_1(\boldsymbol{e}_i)$ is unbounded, then an ideal point does not exist. On the other hand, if a program $P_1(\boldsymbol{e}_i)$ happens to be infeasible, then the MOLP is infeasible.

If the above initialization succeeds, we iterate the following process: we find a point on the boundary of $\mathcal{P}$ and construct a supporting hyperplane in this boundary point. To achieve this goal, the algorithm utilizes the linear programming oracle problem suggested by Benson [11] and its dual:

$$
\begin{array}{lll}
P_2(y)\,\min & z & \\
& A\boldsymbol{x} & \geq \boldsymbol{b} \\
& C\boldsymbol{x} - \mathbf{1}^T z & \leq \boldsymbol{y} \\
& (\boldsymbol{x}, \quad z) & \in \mathbb{R}^{n+1}
\end{array}
$$

and

$$
\begin{array}{lll}
D_2(y)\ \max & \boldsymbol{b}^T\boldsymbol{u} - \boldsymbol{y}^T\boldsymbol{\ell} & \\
& A^T\boldsymbol{u} - C^T\boldsymbol{\ell} & = \mathbf{0} \\
& ||\boldsymbol{\ell}||_1 & = 1 \\
& (\boldsymbol{u}, \quad \boldsymbol{\ell}) & \in \mathbb{R}^{m+d}_{\geq}
\end{array}
$$

In each iteration, we pick an extreme point $\boldsymbol{v}$ of L which we have not yet stored to the output R; if there is no such extreme point, we are done (cf. Line 7). From this point $\boldsymbol{v}$, we construct a point on the boundary of the upper image $\mathcal{P}$: We shoot a ray starting in $\boldsymbol{v}$ in the direction of $\mathbf{1}$ and find the first point of $\mathcal{P}$ the ray hits. It may happen, that $\boldsymbol{v}$ itself is a point on the boundary of $\mathcal{P}$, but then we have proven that $\boldsymbol{v}$ is an extreme point of $\mathcal{P}$. To achieve this, we compute a solution $(\boldsymbol{x}, z)$ of $P_2(\boldsymbol{v})$ (following Hamel, Löhne, and Rudloff [73], cf. Line 8). If $z = 0$, then $\boldsymbol{v}$ is a vertex of $\mathcal{P}$ and for the solution $\boldsymbol{x}$ of P_2 we have that $C\boldsymbol{x} = \boldsymbol{v}$. Thus, the algorithm adds $(\boldsymbol{x}, \boldsymbol{v})$ to the output set R (cf. Line 10).

In the case $z > 0$, $\boldsymbol{v}$ is not in $\mathcal{P}$, but $\boldsymbol{v} + z\mathbf{1}$ is on the boundary of $\mathcal{P}$. The algorithm now has to find a new supporting hyperplane to a face F of $\mathcal{P}$ where $\boldsymbol{v} + z\mathbf{1} \in F$. This can be accomplished by finding an optimal solution $(\boldsymbol{u}, \boldsymbol{\ell})$ of $D_2(\boldsymbol{v} + z\mathbf{1})$ (cf. Line 12).

The hyperplane $\{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{\ell}^T \boldsymbol{x} = b^T \boldsymbol{u}\}$ is a supporting hyperplane to such a face of $\mathcal{P}$. In the end of each iteration in which we find a new supporting hyperplane, we need to recompute the extreme points of L (cf. Line 14).

Listing 1 Benson's Outer Approximation Algorithm

Require: Matrices A, C, and vector $\boldsymbol{b} : P \neq \emptyset$ and $\exists \boldsymbol{y} \in \mathbb{Q}^d : \boldsymbol{y} + C \cdot P \subseteq \mathbb{R}^d_{\geq}$
Ensure: List R of pairs $(\boldsymbol{x}, \boldsymbol{y})$ for all $\boldsymbol{y} \in \operatorname{vert} \mathcal{P}$ and some $\boldsymbol{x} \in P$ such that $C\boldsymbol{x} = \boldsymbol{y}$

1: **for** $i \in [d]$ **do**
2: $\quad (\boldsymbol{y}_{\min})_i \leftarrow$ optimal solution value of $P_1(\boldsymbol{e}_i)$
3: $L \leftarrow \{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{x}_i \geq (\boldsymbol{y}_{\min})_i \text{ for } i \in [n]\}$ ▷ outer approximate polyhedron
4: $N \leftarrow M \leftarrow \{\boldsymbol{y}_{\min}\}$ ▷ extreme points of L
5: $R \leftarrow \emptyset$ ▷ set to be returned
6: **while** $N \neq \emptyset$ **do** ▷ Exists new extreme points
7: $\quad$ pick a $\boldsymbol{v} \in N$
8: $\quad (\boldsymbol{x}, z) \leftarrow$ optimal solution of $P_2(\boldsymbol{v})$
9: $\quad$ **if** $z = 0$ **then**
10: $\quad\quad R \leftarrow R \cup \{(\boldsymbol{x}, \boldsymbol{v})\}$
11: $\quad$ **else**
12: $\quad\quad (\boldsymbol{u}, \boldsymbol{\ell}) \leftarrow$ optimal solution of $D_2(\boldsymbol{v} + z\mathbf{1})$
13: $\quad\quad L \leftarrow L \cap \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{\ell}^T \boldsymbol{x} \leq b^T u\}$
14: $\quad\quad M \leftarrow$ extreme points of L ▷ vertex enumeration
15: $\quad\quad N \leftarrow \{\boldsymbol{y}' \in M \mid \forall (\boldsymbol{x}, \boldsymbol{y}) \in R : \boldsymbol{y} \neq \boldsymbol{y}'\}$

Proofs of correctness of this presentation of Benson's algorithm can be found in the works by Ehrgott, Löhne, and Shao [59] and Hamel, Löhne, and Rudloff [73].

5.2.1. Running Time Analysis of the Original Algorithm

Since 1988 (cf. Ruhe [119]) it is known that the upper image of an MOLP can have an exponential number of extreme points and facets in the dimensions of C and A, so the running time cannot be polynomial and a classical running time analysis was long deemed uninteresting. We now conduct a running time analysis in the framework of output-sensitive complexity. Consequently, we see that there is potential to improve the algorithm to get a better running time bound; this also enables us to show that the improved algorithm achieves incremental polynomial delay.

To investigate the running time of the original algorithm, we first need to be able to say something about the number of iterations: In each iteration we find either a new face supporting inequality or a new extreme point. Hence, the number of iterations is bounded by the number of faces of $\mathcal{P}$. If the number of extreme points is our measure of the output length, we need to know how many faces the upper image has. To bound the number of faces of $\mathcal{P}$, we prove the following lemma.

Lemma 5.1. *Let v_e be the number of extreme points and v_f be the number of faces of $\mathcal{P}$. Then we have $v_f \in \mathcal{O}((v_e(d+1))^{\lfloor \frac{d}{2} \rfloor})$.*

Proof. The polyhedron $\mathcal{P}$ has d extreme recession directions (cf. Burton and Özlen [27]). If we bound it by a halfspace $\{\boldsymbol{x} \in \mathbb{R}^d \mid ||\boldsymbol{x}||_1 \leq \alpha\}$ for a large $\alpha > 1$, we introduce at most dv_e new extreme points. Thus, we can bound the number of faces of the polyhedron $\mathcal{P}$ by $\mathcal{O}(((d+1)v_e)^{\lfloor \frac{d}{2} \rfloor})$ using the asymptotic upper bound theorem by Seidel [123]. □

By a duality argument for the polytope in the above proof, we can also bound the number of faces of $\mathcal{P}$ in the number of facets $v_F : v_f \in \mathcal{O}((v_F(d+1))^{\lfloor \frac{d}{2} \rfloor})$.

In Line 14 of Listing 1, we need to execute a vertex enumeration of the system of linear inequalities L. Since we assume the number of objective functions to be a fixed number, the best choice for this task is the algorithm by B. Chazelle [30]. This algorithm has a running time of $\mathcal{O}(n^{\lfloor \frac{d}{2} \rfloor} + n \log n)$ for a system of linear inequalities of size n in $\mathbb{R}^d$ and is asymptotically optimal for fixed d.

In each iteration, we solve problem P_2 and in each iteration where $\boldsymbol{v}$ is not an extreme point, we also solve problem D_2, additionally, we have problem P_1 which needs to be solved in the initializations. While we can solve P_1 by, e.g., the ellipsoid method (cf. Grötschel, Lovász, and Schrijver [70]) or the inner point method by N. Karmakar [81], it is not so obvious with problems D_2 and P_2 for the following reason: Looking closer into problem P_2, the input consists of data from the input of the MOLP, i.e., A, C and $\boldsymbol{b}$, but also of the vector $\boldsymbol{y}$ which is recomputed while the algorithm runs. Now, the ellipsoid method and also the inner-point method are sensitive to the encoding length of the numbers in the input. Since $\boldsymbol{y}$ is recomputed while the algorithm runs, it is not clear how the encoding length of the $\boldsymbol{y}$s grow in the process of the algorithm. Even if we show that it grows only by a polynomial x^k in an iteration, in the next iteration it can grow again by this polynomial and the new length is x^{2k}, concluding in an exponential size in the number of iterations. Thus, let us assume we have oracles which solve these problems for us: $\mathcal{A}_{P_1}$, $\mathcal{A}_{P_2}$ and $\mathcal{A}_{D_2}$

Theorem 5.2. *Let v_f be the number of faces of $\mathcal{P}$. Then the original or primal Benson Algorithm*

1. *runs oracle $\mathcal{A}_{P_1}$ at most d times,*
2. *runs oracle $\mathcal{A}_{P_2}$ at most $\mathcal{O}(v_f)$ times,*
3. *runs oracle $\mathcal{A}_{D_2}$ at most v_f times,*
4. *performs a vertex enumeration at most v_f times with a running time of $\mathcal{O}(v_f{}^{\lfloor \frac{d}{2} \rfloor} + v_f \log v_f)$, and*
5. *has an additional overhead of $\mathcal{O}(v_f{}^{\lfloor \frac{d}{2} \rfloor} \log v_f)$ for every vertex enumeration.*

when fixing the number of objectives $d \geq 2$.

Proof. 1. To compute $y_{\min}$, we have to solve P_1 d times.

2. We solve $\mathcal{A}_{P_2}$ in each iteration and there are at most $v_f + v_e$ iterations.

3. In each iteration where we do not find an extreme point, we have to run oracle $\mathcal{A}_{P_2}$. These are at most v_f iterations.

4. In each iteration in which we did not find an extreme point, we also conduct a vertex enumeration, i.e., v_f times.

5. Additionally, we have to find the extreme points of L which have not yet been checked (cf. Line 7 in Listing 1). Suppose we computed the set M of extreme points by the vertex enumeration procedure in the last iteration and our current output set is R. We see that $R \subseteq M$. To find a point $v \in M \backslash R$, we can sort both sets and find the first index were the sequences differ. By the upper bound theorem we know that $|M| \in \mathcal{O}(v_f{}^{\lfloor \frac{d}{2} \rfloor})$, as the number of inequalities which define L is at most the number of faces of $\mathcal{P}$. Thus sorting takes time $\mathcal{O}(v_f{}^{\lfloor \frac{d}{2} \rfloor} \log v_f)$ and we do this every time we perform a vertex enumeration.

□

Now we can investigate two viewpoints: While the formulation of the algorithm above is centered around finding the extreme points of $\mathcal{P}$, the algorithm can also be used to find the facets of $\mathcal{P}$, so it can solve MOLP^{YEx} and MOLP^{YFa}. Note though that since we enumerate faces and not facets of $\mathcal{P}$, we need to make sure to output facets only by removing the redundant face supporting inequalities before the algorithm terminates which can be done in time $\text{poly}(d, N, n)$ [80]. To arrive at output-sensitive running times, we need to solve one remaining problem, though: The running times in Theorem 5.2 are bounded by the number of faces and not the facets or extreme points. By employing Lemma 5.1 we can observe the following running times:

Corollary 5.3. *Let N be the number of extreme points (facets) of $\mathcal{P}$. Then the original or primal Benson Algorithm*

1. *runs oracle $\mathcal{A}_{P_1}$ at most d times,*
2. *runs oracle $\mathcal{A}_{P_2}$ at most $\mathcal{O}(N^{\lfloor \frac{d}{2} \rfloor})$ times,*
3. *runs oracle $\mathcal{A}_{D_2}$ at most $\mathcal{O}(N^{\lfloor \frac{d}{2} \rfloor})$ times,*
4. *performs a vertex enumeration at most $\mathcal{O}(N^{\lfloor \frac{d}{2} \rfloor})$ times at a running time of $\mathcal{O}(N^{\frac{d^2}{4}} + N^{\lfloor \frac{d}{2} \rfloor} \log N)$, and*
5. *has an additional overhead of $\mathcal{O}(N^{\frac{d^2}{4} + \lfloor \frac{d}{2} \rfloor} \log N)$.*

when fixing the number of objectives $d \geq 2$.

As an immediate consequence of these running times, we see that if we can solve the problems P_1, P_2 and D_2 in strongly polynomial time for a set of instances, the algorithm runs in output-polynomial time time on these instances for each fixed d. So we arrive at this central corollary for the original version of Benson's algorithm:

Corollary 5.4. *If for an instance of MOLP$^{\text{YEx}}$ or MOLP$^{\text{YFa}}$ we can solve problems P_2 and D_2 in strongly polynomial time and $y_{\min}$ exists, then the original Benson's algorithm is output-sensitive for each fixed $d \geq 2$.*

The obvious next question is if we can bound the delay of the algorithm. To answer it, we need to ask when we know that an extreme point we have found is an extreme point of $\mathcal{P}$ and how much running time went into finding it? We are sure to have found an extreme point in Line 10 in Listing 1, but it is unclear how many faces we enumerated until this point. To answer the same question regarding finding of facets of $\mathcal{P}$, we know that a supporting inequality supports $\mathcal{P}$ in a facet only at the end when we remove redundant inequalities.

One serious drawback of the algorithm is that we might enumerate many redundant supporting hyperplanes, where we only really need the inequalities supporting $\mathcal{P}$ in its facets. This is especially a problem since the number of vertex enumeration steps depends on the number of supporting inequalities we find and also the number of redundant extreme points we compute depends heavily on this quantity. Hence, it is very much desirable to enumerate only facet supporting hyperplanes, which is the subject of the following section.

5.2.2. Finding Facets Exclusively

To ensure finding facet supporting inequalities exclusively, we first add some new insights to the theory of *weight set decompositions* which has been started by Benson and Sun [14] and further developed by Przybylski, Gandibleux, and Ehrgott [114]. We give a characterization of the weight vectors of a member of the nondominated set by means of a hyperplane representation of the polyhedron $\mathcal{P}$.

Weight Space Theory

The *weight set* of a point $\boldsymbol{y} \in \min \mathcal{P}$ is the set

$$W_{\mathcal{P}}(\boldsymbol{y}) := \{\boldsymbol{\ell} \in \mathbb{R}^d \mid \boldsymbol{\ell}^T \boldsymbol{y} \leq \boldsymbol{\ell}^T \boldsymbol{y}', \forall \boldsymbol{y}' \in \mathcal{P}, \boldsymbol{\ell} \geq 0\}.$$

In other words, it is the set of non-negative vectors $\boldsymbol{\ell}$ such that the weighted sum linear program $P_1(\boldsymbol{\ell})$ yields a solution $\boldsymbol{x}$ such that $C\boldsymbol{x} = \boldsymbol{y}$. The *normalized weight set* of such a vector $\boldsymbol{y}$ is then defined as the intersection with the set of *normalized weighting vectors* $W^0 := W^0_d := \{\boldsymbol{\ell} \in \mathbb{R}^d_{\geq} \mid ||\boldsymbol{\ell}||_1 = 1\}$, i.e., $W^0_{\mathcal{P}}(\boldsymbol{y}) := W_{\mathcal{P}}(\boldsymbol{y}) \cap W^0$.

We first show that the general normal cone and the more specialized weight set of $\mathcal{P}$ in a point $\boldsymbol{y} \in \min \mathcal{P}$ coincide except for the sign.

Lemma 5.5. *Let $\boldsymbol{y} \in \min \mathcal{P}$, then $W_{\mathcal{P}}(\boldsymbol{y}) = -N_{\mathcal{P}}(\boldsymbol{y})$.*

Proof. We see that $T_{\mathcal{P}}(\boldsymbol{y}) \supseteq \mathbb{R}^d_{\geq}$. It follows that $T_{\mathcal{P}}(\boldsymbol{y})^* \subseteq \mathbb{R}^{d\,*}_{\geq}$. With $T_{\mathcal{P}}(\boldsymbol{y})^* = N_{\mathcal{P}}(\boldsymbol{y})$ from Theorem 6.28 of Rockafellar and Wets [116] we get that $N_{\mathcal{P}}(\boldsymbol{y}) = T_{\mathcal{P}}(\boldsymbol{y})^* \subseteq$

$\mathbb{R}^{d\,*}_{\geq} = \mathbb{R}^d_{\leq}$. Then

$$\begin{aligned} W_{\mathcal{P}}(y) &= \{\ell \in \mathbb{R}^d \mid y^T\ell \leq \hat{y}^T\ell, \hat{y} \in \mathcal{P}, \ell \geq 0\} \\ &= -(\{\ell \in \mathbb{R}^d \mid \hat{y}^T\ell \leq y^T\ell, \hat{y} \in \mathcal{P}\} \cap \mathbb{R}^d_{\leqq}) \\ &= -(N_{\mathcal{P}}(y) \cap \mathbb{R}^d_{\leqq}) = -N_{\mathcal{P}}(y). \end{aligned}$$

□

We can now use properties of normal cones to prove the crucial theorem for finding facets in Benson's algorithm. Since $\mathcal{P}$ is a polyhedron, there is a set of k inequalities characterizing it: $\mathcal{P} = \{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{a}_i^T\boldsymbol{x} \leq \alpha_i, i \in [k]\}$. For each point y of the nondominated set, we define the *active index set* $I_{\mathcal{P}}(\boldsymbol{y}) := \{i \in [k] \mid \boldsymbol{a}_i^T\boldsymbol{y} = \alpha_i\}$ for such a characterization.

Then, the following theorem states informally: If we take a weakly nondominated point $\boldsymbol{y} \in \min \mathcal{P}$ then the set $W^0_{\mathcal{P}}(\boldsymbol{y})$ is exactly the set of all convex combinations of normalizations of the inequalities which are active in $\boldsymbol{y}$, i.e., satisfy $\boldsymbol{a}_i^T\boldsymbol{y} = \alpha_i$.

Theorem 5.6. $\boldsymbol{y} \in \min \mathcal{P} \implies W^0_{\mathcal{P}}(\boldsymbol{y}) = \operatorname{conv}\{-||\boldsymbol{a}_i||_1^{-1}\boldsymbol{a}_i \mid i \in I_{\mathcal{P}}(\boldsymbol{y})\}$

Proof. From Theorem 6.46 from Rockafellar and Wets [116] and Lemma 5.5 we get that $W_{\mathcal{P}}(\boldsymbol{y}) = -N_Y(\boldsymbol{y}) = \operatorname{cone}\{-\boldsymbol{a}_i \mid i \in I_{\mathcal{P}}(\boldsymbol{y})\}$. Then, we can intersect this set with the normalized weight vectors W^0:

$$\begin{aligned} W^0_{\mathcal{P}}(\boldsymbol{y}) &= W_{\mathcal{P}}(\boldsymbol{y}) \cap W^0 = \operatorname{cone}\{-\boldsymbol{a}_i \mid i \in I_{\mathcal{P}}(\boldsymbol{y})\} \cap W^0 \\ &= \{-\sum_{i \in I_{\mathcal{P}}(\boldsymbol{y})} \gamma_i \boldsymbol{a}_i \mid i \in I_{\mathcal{P}}(\boldsymbol{y}) : \gamma_i \geq 0, \sum_{j \in [d]} (-\sum_{i \in I_{\mathcal{P}}(\boldsymbol{y})} \gamma_i \boldsymbol{a}_i)_j = 1\} \end{aligned}$$

Instead of considering the $\boldsymbol{a}_i$, we can normalize them yielding the same conical combination. So the above set expands to

$$\{-\sum_{i \in I_{\mathcal{P}}(\boldsymbol{y})} \gamma_i ||\boldsymbol{a}_i||_1^{-1}\boldsymbol{a}_i \mid i \in I_{\mathcal{P}}(\boldsymbol{y}) : \gamma_i \geq 0, \sum_{j \in [d]} (-\sum_{i \in I_{\mathcal{P}}(\boldsymbol{y})} \gamma_i ||\boldsymbol{a}_i||_1^{-1}\boldsymbol{a}_i)_j = 1\}.$$

Since $N_{\mathcal{P}} \subseteq \mathbb{R}^d_{\geq}$ and using Theorem 6.46 of Rockafellar and Wets [116] again, we see that no row vector $\boldsymbol{a}_i$ has positive components. A close look at the normalization part thus unveils

$$\begin{aligned} \sum_{j \in [d]} (-\sum_{i \in I_{\mathcal{P}}(\boldsymbol{y})} \gamma_i ||\boldsymbol{a}_i||_1^{-1}\boldsymbol{a}_i)_j &= \sum_{i \in I_{\mathcal{P}}(\boldsymbol{y})} \gamma_i ||\boldsymbol{a}_i||_1^{-1} \sum_{j \in [d]} -(\boldsymbol{a}_i)_j \\ &= \sum_{i \in I_{\mathcal{P}}(\boldsymbol{y})} \gamma_i ||\boldsymbol{a}_i||_1^{-1} ||\boldsymbol{a}_i||_1 = \sum_{i \in I_{\mathcal{P}}(\boldsymbol{y})} \gamma_i \end{aligned}$$

Then we can rewrite $W^0_{\mathcal{P}}(\boldsymbol{y})$ as

$$\begin{aligned} W^0_{\mathcal{P}}(\boldsymbol{y}) &= \{-\sum_{i \in I_{\mathcal{P}}(\boldsymbol{y})} \gamma_i ||\boldsymbol{a}_i||_1^{-1}\boldsymbol{a}_i \mid i \in I_{\mathcal{P}}(\boldsymbol{y}) : \gamma_i \geq 0, \sum_{i \in I_{\mathcal{P}}(\boldsymbol{y})} \gamma_i = 1\} \\ &= \operatorname{conv}\{-||\boldsymbol{a}_i||_1^{-1}\boldsymbol{a}_i \mid i \in I_{\mathcal{P}}(\boldsymbol{y})\}. \end{aligned}$$

□

The above theorem allows us to draw a connection between the weight set of a point $y \in \min \mathcal{P}$ and the facets of $\mathcal{P}$ in which y lies.

Finding Facets

We now show how we can ensure to find facets only in Benson's algorithm. The key ingredient is that we can always find a facet supporting inequality in a point $\boldsymbol{y}$ by computing an extreme point of the polytope $W^0_{\mathcal{P}}(\boldsymbol{y})$. This can be proven by fixing a nondegenerate representation of $\mathcal{P}$ and using Theorem 5.6.

Lemma 5.7. *Let $\boldsymbol{y} \in \min \mathcal{P}$ and $\boldsymbol{\ell} \in \operatorname{vert} W^0_{\mathcal{P}}(\boldsymbol{y})$. Then, there is an $\tilde{\alpha} \in \mathbb{R}$ such that $\{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{\ell}^T \boldsymbol{x} = \tilde{\alpha}\}$ supports $\mathcal{P}$ in a facet.*

Proof. We see that $W^0_{\mathcal{P}}(\boldsymbol{y})$ is independent of the inequality representation of $\mathcal{P}$. Let us consider the case where $\boldsymbol{a}_i$ and α_i are part of a nonredundant representation of $\mathcal{P}$.

Theorem 5.6 gives us a characterization of $W^0_{\mathcal{P}}(\boldsymbol{y}) = \operatorname{conv}\{-||\boldsymbol{a}_i||_1^{-1}\boldsymbol{a}_i \mid i \in I_{\mathcal{P}}(\boldsymbol{y})\}$. Since $\boldsymbol{\ell}$ is a vertex of $W^0_{\mathcal{P}}(\boldsymbol{y})$, there exists an $\hat{i} \in I_{\mathcal{P}}(\boldsymbol{y})$ such that $\boldsymbol{\ell} = -||\boldsymbol{a}_{\hat{i}}||_1^{-1}\boldsymbol{a}_{\hat{i}}$. Using the nonredundancy, we have that the hyperplane

$$\{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{a}_{\hat{i}}^T \boldsymbol{x} = \alpha_{\hat{i}}\} = \{\boldsymbol{x} \in \mathbb{R}^d \mid -||\boldsymbol{a}_{\hat{i}}||_1^{-1}\boldsymbol{a}_{\hat{i}}^T \boldsymbol{x} = -||\boldsymbol{a}_{\hat{i}}||_1^{-1}\alpha_{\hat{i}}\}$$

supports $\mathcal{P}$ in a facet. Thus, setting $\tilde{\alpha} := -||\boldsymbol{a}_{\hat{i}}||_1^{-1}\alpha_{\hat{i}}$ proves the claim. □

The question how to compute such an extreme point focuses on the oracle problem D_2. Up to now, we were looking for any $\boldsymbol{\ell}$ which was part of a solution to D_2. We show that the projection on the $\boldsymbol{\ell}$ variables of the optimal solutions to $D_2(\boldsymbol{y})$ is exactly the normalized weight set of $\boldsymbol{y} \in \min \mathcal{P}$.

Lemma 5.8. $\boldsymbol{y} \in \min \mathcal{P} \implies W^0_{\mathcal{P}}(\boldsymbol{y}) = \operatorname{proj}_{\boldsymbol{\ell}}(\arg\max D_2(\boldsymbol{y}))$

Proof. Let us first show that $W^0_{\mathcal{P}}(\boldsymbol{y}) \supseteq \operatorname{proj}_{\boldsymbol{\ell}}(\arg\max D_2(\boldsymbol{y}))$. Theorem 2.5 of Benson [11] concludes, that $W_{\mathcal{P}}(\boldsymbol{y}) \supseteq \operatorname{proj}_{\boldsymbol{\ell}}(\arg\max D_2(\boldsymbol{y}))$, but since for every $\boldsymbol{\ell}$ which is part of a feasible solution to $D_2(\boldsymbol{y})$ the equation $||\boldsymbol{\ell}||_1 = 1$ holds, so does $W^0_{\mathcal{P}}(\boldsymbol{y}) \supseteq \operatorname{proj}_{\boldsymbol{\ell}}(\arg\max D_2(\boldsymbol{y}))$.

Regarding $W^0_{\mathcal{P}}(\boldsymbol{y}) \subseteq \operatorname{proj}_{\boldsymbol{\ell}}(\arg\max D_2(\boldsymbol{y}))$, by definition

$$\operatorname{proj}_{\boldsymbol{\ell}}(\arg\max D_2(\boldsymbol{y})) = \{\boldsymbol{\ell} \in \mathbb{R}^d_{\geq} \mid \boldsymbol{u} \in \mathbb{R}^m_{\geq}, b^T\boldsymbol{u} = \boldsymbol{y}^T\boldsymbol{\ell}, \mathbf{1}^T\boldsymbol{\ell} = 1, A^T\boldsymbol{u} = C^T\boldsymbol{\ell}\}.$$

Let $\boldsymbol{y} \in \min \mathcal{P}$ and $\boldsymbol{\ell} \in W^0_{\mathcal{P}}(\boldsymbol{y})$. Following the definition of $W^0_{\mathcal{P}}(\boldsymbol{y})$ we get $\boldsymbol{\ell}^T\boldsymbol{y} \leq \boldsymbol{\ell}^T\hat{\boldsymbol{y}}$ for $\hat{\boldsymbol{y}} \in \mathcal{P}$. Because $\boldsymbol{y} \in \mathcal{P}$, there is a feasible solution $\boldsymbol{x}$ of P such that $C\boldsymbol{x} = \boldsymbol{y}$. By $\boldsymbol{\ell}^T C\boldsymbol{x} \leq \boldsymbol{\ell}^T C\hat{\boldsymbol{x}}$ for every feasible solution $\hat{\boldsymbol{x}}$ of P, we conclude that $\boldsymbol{x}$ is an optimal solution to $P_1(\boldsymbol{\ell})$.

Let us denote the dual problem to $P_1(\boldsymbol{\ell})$ by $D_1(\boldsymbol{\ell})$ which thus is

$$D_1(\boldsymbol{\ell}) : \max\{b^T\boldsymbol{u} \mid \boldsymbol{u} \in \mathbb{R}^m_{\geq}, A^T\boldsymbol{u} = C^T\boldsymbol{\ell}\}.$$

In the above case, we can find a vector $\boldsymbol{u} \in \mathbb{R}^m_{\geq}$ which is optimal for $D_1(\boldsymbol{\ell})$ because of duality of these linear programs. Then, we have $A^T\boldsymbol{u} = C^T\boldsymbol{\ell}$ and $b^T\boldsymbol{u} = \boldsymbol{\ell}^T C\boldsymbol{x} = \boldsymbol{\ell}^T\boldsymbol{y}$. □

We can conclude that an extreme point of $W_{\mathcal{P}}^0(\boldsymbol{y})$ can be computed by finding an extreme point solution of the solutions of $D_2(\boldsymbol{y})$ projected on the $\boldsymbol{\ell}$-variables. This is not necessarily an extreme point solution of $D_2(\boldsymbol{y})$ itself. We now prove a Lemma, which allows us to reduce this problem to a lexicographic linear optimization problem.

Lemma 5.9. *If a point $\boldsymbol{y}$ in an n-polyhedron $P = \{\boldsymbol{x} \in \mathbb{R}^d \mid A\boldsymbol{x} \leq \boldsymbol{b}\}$ is lexicographically minimal in P then $\boldsymbol{y}$ is an extreme point of P.*

Proof. Let us assume that $\boldsymbol{y}$ is lexicographically minimal in P and $\boldsymbol{y}$ is not an extreme point of P. Then $\boldsymbol{y}$ is in the relative interior of some face of P. Let F be the unique face containing $\boldsymbol{y}$ in its relative interior. (Note that P itself is a face of P by definition.) Since $\boldsymbol{y} \in \operatorname{ri} F$, there exists an $\varepsilon > 0$ such that $B := B_\varepsilon(\boldsymbol{y}) \cap \operatorname{aff} F \subseteq F$.

Then we can construct a lexicographically smaller point in P to create a contradiction: We take a normalized affine base $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k\}$ of $\operatorname{aff} F$ with $||\boldsymbol{v}_i||_2 = 1$ for every $i \in [k]$ and pick one of the base vectors $\boldsymbol{v}$. One of the points $\boldsymbol{y}+\frac{\varepsilon}{2}\boldsymbol{v}$ or $\boldsymbol{y}-\frac{\varepsilon}{2}\boldsymbol{v}$ is lexicographically smaller than $\boldsymbol{y}$ itself. We also observe that both points are in B and thus in F. □

This shows that we can easily compute an extreme point of $W_{\mathcal{P}}^0(\boldsymbol{y})$ by solving the lexicographic optimization problem

$$\text{lex-}D_2(\boldsymbol{y}) : \operatorname{lexmin}\{(b^T\boldsymbol{u} - \boldsymbol{\ell}^T\boldsymbol{y}, \boldsymbol{\ell}_1, \ldots, \boldsymbol{\ell}_{d-1}) \mid (\boldsymbol{u}, \boldsymbol{\ell}) \in \mathbb{R}^{m+d}_{\geq}, A^T\boldsymbol{u} = C^T\boldsymbol{\ell}, ||\boldsymbol{\ell}||_1 = 1\}.$$

To implement this improvement into the algorithm, we simply have to change line 12 in Algorithm 1 accordingly.

Running Time Analysis

Now let us state the running times of the new, improved algorithm. And now we also have an oracle $\mathcal{A}_{\text{lex-}D_2}$ for solving problem lex-D_2.

Proposition 5.10. *Let v_e (v_F) be the number of extreme points (facets) of $\mathcal{P}$. Then the improved version of Benson's Algorithm*

1. *runs oracle $\mathcal{A}_{P_1}$ at most d times,*
2. *runs oracle $\mathcal{A}_{P_2}$ at most $v_e + v_F$ times,*
3. *runs oracle $\mathcal{A}_{lex\text{-}D_2}$ at most v_F times,*
4. *performs a vertex enumeration at most v_F times with a running time of $\mathcal{O}({v_F}^{\lfloor \frac{d}{2} \rfloor} + v_F \log v_F)$, and*
5. *has an additional overhead of $\mathcal{O}({v_F}^{\lfloor \frac{d}{2} \rfloor} \log v_F)$ per vertex enumeration.*

when fixing the number of objectives $d \geq 2$.

We see that not much has actually changed in the view of problem MOLP^{YEx}. The best bound for v_F in the number of extreme points still is the more general bound of Lemma 5.1 and so we arrive at the same running times as in Theorem 5.3. But in the light of MOLP^{YFa} a lot has changed: Besides (2) in the above theorem, we only deal with the number of facets. Let us state this as a first corollary.

Corollary 5.11. *Let v_F be the number of facets of $\mathcal{P}$. Then the improved version of Benson's Algorithm*

1. *runs oracle $\mathcal{A}_{P_1}$ at most d times,*
2. *runs oracle $\mathcal{A}_{P_2}$ at most $\mathcal{O}({v_F}^{\lfloor \frac{d}{2} \rfloor})$ times,*
3. *runs oracle $\mathcal{A}_{lex\text{-}D_2}$ at most v_F times,*
4. *performs a vertex enumeration at most v_F times with a running time of $\mathcal{O}({v_F}^{\lfloor \frac{d}{2} \rfloor} + v_F \log v_F)$, and*
5. *has an additional overhead of $\mathcal{O}({v_F}^{\lfloor \frac{d}{2} \rfloor + 1} \log v_F)$.*

when fixing the number of objectives $d \geq 2$.

We see that we improved the running time and were able to get rid of the d^2 in the exponent for problem MOLP^{YFa}. Moreover, we can make another observation: The intermediate extreme points $\boldsymbol{y}$ of the linear inequality system L come from facet defining inequalities of $\mathcal{P}$ instead of face supporting inequalities. The difference is, that we can relate the facet defining inequalities directly to the input encoding and are now able to bound the encoding length of the points $\boldsymbol{y}$. To prove this, we base the following observation on Lemma 1.3.4 by Grötschel, Lovász, and Schrijver [70] on the encoding length of the determinant of a matrix and using Cramer's rule.

Lemma 5.12 (Grötschel, Lovász, and Schrijver [70])**.** *If a rational linear system $A\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x} \in \mathbb{R}^n$ with $A \in \mathbb{Q}^{m \times n}$ and $\boldsymbol{b} \in \mathbb{Q}^m$ has a unique solution $\boldsymbol{x}^*$, then $\boldsymbol{x}^* \in \mathbb{Q}^n$ and $\langle \boldsymbol{x}_i^* \rangle \in \mathcal{O}(\text{poly}(\langle A \rangle, \langle \boldsymbol{b} \rangle))$.*

The first step is to bound the encoding length of any extreme point of $\mathcal{P}$. From this we construct a supporting inequality to a given facet and prove that the encoding length of the coefficients of this inequality grow only polynomially in the input size. By normalization we arrive at the inequalities we compute by solving lex-D_2 and we show that the numbers do not grow too fast if we normalize the vector. So let's first investigate extreme points of $\mathcal{P}$.

Lemma 5.13. *For every extreme point $\boldsymbol{y}$ of $\mathcal{P}$ it we have $\langle \boldsymbol{y}_i \rangle \in \mathcal{O}(\text{poly}(\langle C \rangle, \langle A \rangle, \langle \boldsymbol{b} \rangle))$.*

Proof. For an extreme point $\boldsymbol{y}$ in $\mathcal{P}$ there exists an extreme point $\boldsymbol{x}^*$ of $P = \{\boldsymbol{x} \in \mathbb{R}^n \mid A\boldsymbol{x} \leq \boldsymbol{b}\}$ such that $C\boldsymbol{x}^* = \boldsymbol{y}$. We can write $\boldsymbol{y}$ as the unique solution of the following

rational linear system containing only numbers of the input.

$$\begin{aligned} C\boldsymbol{x} &= \boldsymbol{y} \\ \boldsymbol{a}_i{}^T\boldsymbol{x} &= b_i \text{ for } i \in I(\boldsymbol{x}^*) \\ \boldsymbol{x} &\in \mathbb{R}^n \end{aligned}$$

□

Departing from the extreme points, we use some extreme points of a facet of $\mathcal{P}$ to construct a facet supporting inequality with well behaved coefficients.

Lemma 5.14. *For every facet F of $\mathcal{P}$, there exists a facet supporting inequality $\boldsymbol{c}^T\boldsymbol{x} \leq \gamma$ for which $\langle \boldsymbol{c}_i \rangle \in \mathcal{O}(\text{poly}(\langle C \rangle, \langle A \rangle, \langle \boldsymbol{b} \rangle))$ and $\langle \gamma \rangle \in \mathcal{O}(\text{poly}(\langle C \rangle, \langle A \rangle, \langle \boldsymbol{b} \rangle))$.*

Proof. For a facet F of $\mathcal{P}$ there are at least d extreme points $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_d$ of $\mathcal{P}$ that are also extreme points of F. We can find a facet supporting inequality by solving the following linear system:

$$\boldsymbol{y}_i{}^T\boldsymbol{x} = 0, \text{ for } i \in [d] \tag{5.1}$$

$$\mathbf{1}^T\boldsymbol{x} = 1 \tag{5.2}$$

$$\boldsymbol{x} \in \mathbb{R}^d \tag{5.3}$$

The equalities (5.1) describe the linear space of all vectors orthogonal to F. The equality (5.2) ensures that $||\boldsymbol{x}||_1 = 1$ because $\operatorname{rec}\mathcal{P} \subseteq \mathbb{R}^n_{\geq}$ and thus for each facet normal we have that either all components are positive or negative (or $\mathbf{0}$). Since the encoding length of the extreme points $\boldsymbol{y}_i$ can be bounded by a polynomial in the input size, the encoding length of the unique solution $\boldsymbol{c}$ to the above system can also be bounded by a polynomial in the input size. Now we can find γ by solving $\min\{\boldsymbol{c}^T\boldsymbol{x} \mid A\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \in \mathbb{R}^n\}$. Again, as all numbers in this weighted-sum LP are either from the input or can be bounded by a polynomial in the input size, the optimal value can also be bounded by a polynomial in the input size: There is an optimal solution $\boldsymbol{x}^*$ to this problem which is an extreme point of $\{\boldsymbol{x} \in \mathbb{R}^n \mid A\boldsymbol{x} \leq \boldsymbol{b}\}$ and thus $\langle \boldsymbol{x}^* \rangle \in \text{poly}(\langle A \rangle, \langle \boldsymbol{b} \rangle)$. Since $\gamma = \boldsymbol{c}^T\boldsymbol{x}^*$, $\langle \gamma \rangle$ can also be bounded by a polynomial in $\langle A \rangle, \langle C \rangle$ and $\langle \boldsymbol{b} \rangle$. □

Now we can relate these facet defining inequalities to the inequalities we find in the process of the algorithm.

Proposition 5.15. *For an inequality $\boldsymbol{\ell}^T\boldsymbol{x} \leq \boldsymbol{b}^T\boldsymbol{u}$ we compute by solving D_2 in the improved algorithm, it holds that $\langle \boldsymbol{\ell}_i \rangle \in \mathcal{O}(\text{poly}(\langle C \rangle, \langle A \rangle, \langle \boldsymbol{b} \rangle))$ and $\langle \boldsymbol{b}^T\boldsymbol{u} \rangle \in \mathcal{O}(\text{poly}(\langle C \rangle, \langle A \rangle, \langle \boldsymbol{b} \rangle))$.*

Proof. Let $\boldsymbol{\ell}^T\boldsymbol{x} \leq \boldsymbol{b}^T\boldsymbol{u}$ as computed in the algorithm and supporting a facet F of $\mathcal{P}$. By Lemma 5.14, there also exists $\boldsymbol{c} \in \mathbb{Q}^d$ and $\gamma \in \mathbb{Q}$ such that $\boldsymbol{c}^T\boldsymbol{x} \leq \gamma$ also supports F. Now, since the facet supporting hyperplane is unique we know that

$$\{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{\ell}^T\boldsymbol{x} \leq \boldsymbol{b}^T\boldsymbol{u}\} = \{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{c}^T\boldsymbol{x} \leq \gamma\}.$$

By construction of $\boldsymbol{\ell}$, we also know that $||\boldsymbol{\ell}||_1 = 1$, but this is also the case for the $\boldsymbol{c}$ constructed in Lemma 5.14. Thus, $\boldsymbol{\ell} = \boldsymbol{c}$ and $\gamma = \boldsymbol{b}^T\boldsymbol{u}$. □

Now let's investigate the intermediate extreme points $\boldsymbol{v}$ we compute in the algorithm: These points are extreme points of the intermediate outer approximation $L = \{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{\ell}_i^T\boldsymbol{x} \leq \boldsymbol{b}^T\boldsymbol{u}_i, i \in [\#\text{iterations}]\}$ of $\mathcal{P}$. We can again write $\boldsymbol{v}$ as the unique solution to the linear system:

$$\begin{aligned} \boldsymbol{\ell}^T\boldsymbol{x} &= \boldsymbol{b}^T\boldsymbol{u}_i, i \in I_L(\boldsymbol{y}) \\ \boldsymbol{x} &\in \mathbb{R}^d \end{aligned}$$

Hence, the vectors $\boldsymbol{v}$ we use in problem P_2 in the improved algorithm can always be bounded by a polynomial in the input size. In contrast to the original algorithm, we now can solve P_2 by the ellipsoid method and arrive at a polynomial running time in the input size independent of the number of iterations. We can also bound the input of the problem lex-D_2: The input for lex-D_2 consists only of numbers from the input and $\boldsymbol{v} + z\boldsymbol{1}$, where $\boldsymbol{v}$ is as described above. Now, z is part of an optimal extreme point solution to P_2 and its encoding length is thus also in $\text{poly}(\langle A\rangle, \langle C\rangle, \langle \boldsymbol{b}\rangle)$. There is one caveat though: lex-D_2 is a lexicographic linear programming problem, but we can show that we can reduce a lexicographic linear programming problem to a series of linear programming problems and prove the following Lemma:

Lemma 5.16. *An optimal solution to a lexicographic LP can be computed in polynomial time.*

Proof. We can prove this by using a well known approach that has—to the best of our knowledge—not been analyzed as yet. We solve the lex-LP $\text{lexmin}\{(\boldsymbol{c}_1{}^T\boldsymbol{x}, \ldots, \boldsymbol{c}_d{}^T\boldsymbol{x}) \mid A\boldsymbol{x} \leq \boldsymbol{b}\}$ by solving a series of single objective LPs. Thus, we define LP_1, …, LP_d, where $\text{LP}_1 : \min\{\boldsymbol{c}_1{}^T\boldsymbol{x} \mid A\boldsymbol{x} \leq \boldsymbol{b}\}$ and $\text{LP}_k : \min\{\boldsymbol{c}_k{}^T\boldsymbol{x} \mid A\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{c}_1{}^T\boldsymbol{x} = y_1, \ldots, \boldsymbol{c}_{k-1}{}^T\boldsymbol{x} = y_{k-1}\}$ for $k \in \{2, \ldots, d\}$ and y_k being the optimal value of LP_k. We see that an optimal solution to LP_d is also an optimal solution to lex-LP.

The question here is how the encoding length of the y_k are growing. By showing that the encoding length of the optimal extreme point solutions of each of the LP_k are always bounded by a polynomial in the input, we see that the encoding length of y_k is always bounded by a polynomial in the input size. But this is apparent by the following observation: Let M_k be the set of optimal extreme point solutions to LP_k, then $M_d \subseteq \cdots \subseteq M_1$. Since the encoding length of the extreme point solutions to LP_1 are bounded by $\text{poly}(\langle A\rangle, \langle \boldsymbol{b}\rangle)$, the claim is proven. □

So in terms of polynomial solvability, by turning D_2 into a lexicographic linear programming problem we did not lose anything. To summarize the observations of this subsection: We showed, that by finding facets of $\mathcal{P}$ exclusively, we get better running times for the problem MOLP^{YFA}, but did not improve on the running time for the problem MOLP^{YEX}. However, we can now get rid of the assumption that P_2 and lex-D_2 need to be solved in strongly polynomial time and we can state the running time in a concrete way:

Theorem 5.17. *Let v_e (v_F) be the number of extreme points (facets) of $\mathcal{P}$. Then the improved version of Benson's algorithm has a running time of*

$$\mathcal{O}((v_e + v_F)\,\mathrm{poly}(\langle A\rangle, \langle C\rangle, \langle \boldsymbol{b}\rangle) + {v_F}^{\lfloor \frac{d}{2}\rfloor+1}\log v_F)$$

for each fixed $d \geq 2$. Moreover, problem MOLP$^{\mathrm{YEx}}$ can be solved in time

$$\mathcal{O}({v_e}^{\lfloor \frac{d}{2}\rfloor}\,\mathrm{poly}(\langle A\rangle, \langle C\rangle, \langle \boldsymbol{b}\rangle) + {v_e}^{\frac{d^2}{4}+\lfloor \frac{d}{2}\rfloor}\log v_e)$$

for each fixed $d \geq 2$ and MOLP$^{\mathrm{YFa}}$ can be solved in time

$$\mathcal{O}({v_F}^{\lfloor \frac{d}{2}\rfloor}\,\mathrm{poly}(\langle A\rangle, \langle C\rangle, \langle \boldsymbol{b}\rangle) + {v_F}^{\lfloor \frac{d}{2}\rfloor+1}\log v_F)$$

for each fixed $d \geq 2$ if $y_{\min}$ exists for all these problems.

And for the problems MOLP$^{\mathrm{YEx}}$ and MOLP$^{\mathrm{YFa}}$ this means:

Corollary 5.18. *If for an instance of MOLP$^{\mathrm{YEx}}$ (MOLP$^{\mathrm{YFa}}$) $y_{\min}$ exists, then the improved version of Benson's algorithm is output-sensitive for solving MOLP$^{\mathrm{YEx}}$ (MOLP$^{\mathrm{YFa}}$) for each fixed $d \geq 2$.*

We can make another observation regarding problem MOLP$^{\mathrm{YFa}}$: Each time we solve problem lex-D_2 we get a facet defining inequality. We can now consider the delay until finding the next facet defining inequality. We have to at most investigate all intermediate extreme points and they are either all extreme points of $\mathcal{P}$ and then we are done, or we find a new facet defining inequality. This way, we can bound the delay of the improved version of Benson's algorithm.

Theorem 5.19. *Let $d \geq 2$ be fixed. The k-th delay of the improved version of Benson's algorithm is*

$$\mathcal{O}(k^{\lfloor \frac{d}{2}\rfloor}(\mathrm{poly}(\langle A\rangle, \langle C\rangle, \langle \boldsymbol{b}\rangle) + \log k))$$

for solving MOLP$^{\mathrm{YFa}}$ with existing $y_{\min}$.

5.3. Finding Extreme Points: The Dual of Benson's Algorithm

We now turn to the Dual Benson Algorithm and discuss its running time. First, we introduce the duality theory in multiobjective linear programming introduced by F. Heyde and A. Löhne in 2008 [75] on which the algorithm is based. Then we describe the algorithm in detail and then turn to its running time analysis.

The Geometric Dual Polyhedron

Heyde and Löhne [75] define a dual polyhedron, or *lower image*, $\mathcal{D}$ to the upper image $\mathcal{P}$ of an MOLP which we define now.

Since weight vectors $\boldsymbol{\ell} \in W_d^0$ of the weighted-sum problem are always normalized in this work, i.e., $||\boldsymbol{\ell}||_1 = 1$, it suffices to consider $\boldsymbol{\ell}_1, \ldots, \boldsymbol{\ell}_{d-1}$ and calculate $\boldsymbol{\ell}_d$ when needed. For ease of notation we define for any $\boldsymbol{v} \in \mathbb{R}^d : \lambda(\boldsymbol{v}) := (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{d-1}, 1 - \sum_{i=1}^{d-1} \boldsymbol{v}_i)$. Then, we consider the dual problem of the weighted-sum LP, which is

$$\begin{aligned} D_1(\boldsymbol{\ell}) \max \ & \boldsymbol{b}^T \boldsymbol{u} \\ & A^T \boldsymbol{u} = C^T \boldsymbol{\ell} \\ & \boldsymbol{u} \in \mathbb{R}^m_{\geq}. \end{aligned}$$

The dual polyhedron $\mathcal{D}$ now consists for all possible vectors $\boldsymbol{\ell} \in W_d^0$ and solutions $\boldsymbol{u}$ to $D_1(\boldsymbol{\ell})$ of the vectors $(\boldsymbol{\ell}_1, \ldots, \boldsymbol{\ell}_{d-1}, \boldsymbol{b}^T \boldsymbol{u})$. Thus,

$$\mathcal{D} := \{(\boldsymbol{\ell}_1, \ldots, \boldsymbol{\ell}_{d-1}, \boldsymbol{b}^T \boldsymbol{u}) \in \mathbb{R}^d \mid \boldsymbol{\ell} \in W_d^0, \boldsymbol{u} \in \mathbb{R}^m_{\geq}, A^T \boldsymbol{u} = C^T \boldsymbol{\ell}\}.$$

Following LP duality theory, for each point $\boldsymbol{y}$ on the upper boundary of this polyhedron $\boldsymbol{y}_d$ is also the optimal value of $P_1(\lambda(\boldsymbol{y}))$. To take the notion of the upper boundary to a more formal level, we define the $\mathcal{K}_d$-maximal subset of a set $M \subseteq \mathbb{R}^d$, where $\mathcal{K}_d := \{(0, \ldots, 0, \alpha) \in \mathbb{R}^d \mid \alpha \geq 0\}$: A vector $\boldsymbol{y} \in M$ is said to be *$\mathcal{K}_d$-maximal* in M if $(\boldsymbol{y} + \mathcal{K}_d) \cap M = \{\boldsymbol{y}\}$. The subset of $\mathcal{K}_d$-maximal points of M is written as $\max_{\mathcal{K}_d} M$.

The dual polyhedron can be characterized by

$$\mathcal{D} = \{\boldsymbol{x} \in \mathbb{R}^d \mid \lambda(\boldsymbol{x}) \geq 0, \forall \boldsymbol{y} \in \min \mathcal{P} : \psi(\boldsymbol{y})^T \boldsymbol{x} \geq -\boldsymbol{y}_d\},$$

where $\psi(\boldsymbol{y}) := (\boldsymbol{y}_1 - \boldsymbol{y}_d, \ldots, \boldsymbol{y}_{d-1} - \boldsymbol{y}_d, -1)$ [75]. Moreover it is proven that only those $\boldsymbol{y} \in \min \mathcal{P}$ are necessary which are extreme points of $\mathcal{P}$. In other words, apart from the inequalities $\lambda(\boldsymbol{x}) \geq 0$, we can describe the polyhedron as an intersection of halfspaces $\{\boldsymbol{x} \in \mathbb{R}^d \mid \psi(\boldsymbol{y})^T \boldsymbol{x} \geq -\boldsymbol{y}_d\}$ for each extreme point $\boldsymbol{y}$ of the upper image $\mathcal{P}$. Further, Heyde and Löhne also prove that each of these inequalities defines a facet. Thus, we can solve MOLP$^{\text{YEx}}$ by enumerating the facets and solve MOLP$^{\text{YFa}}$ by enumerating the extreme points of $\mathcal{D}$. While the dual algorithm originally enumerates the extreme points of $\mathcal{D}$, we change the exposition accordingly to match the description of the primal algorithm.

Algorithm Description

We follow Ehrgott, Löhne, and Shao [59] in describing the geometric dual algorithm and again use ideas by Hamel, Löhne, and Rudloff [73]. Proofs of correctness and finiteness can be found in both places. A description of the entire algorithm in pseudocode can be found in Listing 2.

The algorithm always maintains an outer approximation polyhedron L of $\mathcal{D}$ in an inequality representation. In the initialization, the algorithm constructs a polyhedron

containing $\mathcal{D}$ (Lines 1 to 3). This is done by finding one arbitrary weakly Pareto-optimal solution to the MOLP, in our pseudocode we compute a solution to $P_1(\boldsymbol{e}_1)$ and add the initial inequalities $\lambda(\boldsymbol{x}) \geq 0$.

Then, in each iteration, the algorithm picks one new extreme point $\boldsymbol{v}$ of the current intermediate polyhedron and shoots a ray into the polyhedron $\mathcal{D}$ (Lines 6 and 7). Similar to the original algorithm we want to find out if $\boldsymbol{v}$ already lies on the boundary of $\mathcal{D}$. And similar to the original algorithm, we can do this by shooting a ray into $\mathcal{D}$ starting at $\boldsymbol{v}$. This can be achieved by finding an optimal solution $\boldsymbol{x}$ with value z of $P_1(\lambda(\boldsymbol{v}))$, which is equivalent to shooting a ray in the direction of $-\mathcal{K}_d$.

By doing so, we either discover that $\boldsymbol{v}$ is an extreme point of $\mathcal{D}$ if $z = \lambda(\boldsymbol{v})^T C\boldsymbol{x} = \boldsymbol{v}_d$ and we proceed to the next iteration. Or we discover that v is not an extreme point, which is the case if $z = \lambda(\boldsymbol{v})^T C\boldsymbol{x} < \boldsymbol{v}_d$. In the latter case, the algorithm computes a face defining inequality which separates $\boldsymbol{v}$ from $\mathcal{D}$. Because of geometric duality, we can use the new inequality $\psi(\boldsymbol{y})^T\boldsymbol{x} \geq -\boldsymbol{y}_d$ for $\boldsymbol{y} = C\boldsymbol{x}$. In lines 12 and 13, the algorithm intersects the current polyhedron with the halfspace corresponding to this inequality. Additionally, it saves $\boldsymbol{y}$ as a candidate for an nondominated extreme point in Line 14. This repeats until all extreme points have been confirmed to be part of $\mathcal{D}$. In the end, we still have to remove redundant pairs from the set of candidate nondominated extreme points (see Line 15).

Listing 2 Dual Variant of Benson's Outer Approximation Algorithm

Require: Matrices A, C, and vector $\boldsymbol{b} : P \neq \emptyset$ and $\exists \boldsymbol{y} \in \mathbb{R}^d : \boldsymbol{y} + C \cdot P \subseteq \mathbb{R}^d_{\geq}$
Ensure: List R of pairs $(\boldsymbol{x}, \boldsymbol{y})$ for all $\boldsymbol{y} \in \operatorname{vert}\mathcal{P}$ and some $\boldsymbol{x} \in P$ such that $C\boldsymbol{x} = \boldsymbol{y}$

1: Find solution $\boldsymbol{x}$ of $P_1(\boldsymbol{e}_1)$ and set $\boldsymbol{y} \leftarrow C\boldsymbol{x}$
2: $L \leftarrow \{\boldsymbol{x} \in \mathbb{R}^d \mid \lambda(\boldsymbol{x}) \geq 0, \psi(\boldsymbol{y})^T\boldsymbol{x} \geq -\boldsymbol{y}_d\}$ ▷ Initial polyhedron
3: $M \leftarrow$ Extreme points of L ▷ Perform a vertex enumeration
4: $E \leftarrow \emptyset$ ▷ Extreme points already discovered
5: **while** $M \backslash E \neq \emptyset$ **do**
6: pick one $\boldsymbol{v} \in N$
7: $\boldsymbol{x} \leftarrow$ optimal solution to $P_1(\lambda(\boldsymbol{v}))$ ▷ Shoot ray straight down
8: $\boldsymbol{y} \leftarrow C\boldsymbol{x}$
9: **if** $\lambda(\boldsymbol{v})^T\boldsymbol{y} = \boldsymbol{v}_d$ **then** ▷ Is an extreme point of $\mathcal{D}$
10: $E \leftarrow E \cup \{y\}$
11: **else** ▷ Not an extreme point of $\mathcal{D}$
12: $L \leftarrow L \cap \{\boldsymbol{x} \in \mathbb{R}^d \mid \psi(\boldsymbol{y})^T\boldsymbol{x} \geq -\boldsymbol{y}_d\}$ ▷ Add new inequality
13: $M \leftarrow$Extreme points of L ▷ Perform a vertex enumeration
14: $R \leftarrow R \cup \{(\boldsymbol{x}, \boldsymbol{y})\}$ ▷ Add new candidate extreme point of $\mathcal{P}$
15: Remove redundant entries from R

5.3.1. Running Time Analysis

Let us now investigate the running time of the dual algorithm. Most of the theoretical work we discovered in the previous section on the original algorithm helps us here. In

the end of the analysis we again see, that there is room for improvement and proceed with a subtle change to arrive at a better running time.

Again, the key insight to the running time of the dual algorithm is that the vertex enumeration steps are performed in the ordinarily much smaller domain of the polyhedron $\mathcal{D}$, which is of dimension d. Additionally, the number of inequalities we enumerate in the process of the algorithm is at most the number of $\mathcal{K}$-maximal faces of $\mathcal{D}$ which—by the geometric duality theorem [75]—is exactly the number of faces of $\mathcal{P}$. Thus by Lemma 5.1, the number of faces of $\mathcal{D}$ and thus the number of inequalities the algorithm computes does not exceed $\mathcal{O}(v_e^{\lfloor \frac{d}{2} \rfloor})$ (for every fixed d). To compute the extreme points of the intermediate polyhedra, we can again use the asymptotic optimal algorithm for fixed d by Chazelle [30].

Regarding encoding length, we only solve the weighted sum LPs $P_1(\lambda(\boldsymbol{v}))$. The case is easier here, since the face defining inequalities always come directly from data from the input.

Lemma 5.20. *For an inequality $\psi(\boldsymbol{y})^T\boldsymbol{x} \geq -\boldsymbol{y}_d$ we add in Line 12 of the dual algorithm we have that $\langle \boldsymbol{y} \rangle \in \text{poly}(\langle A \rangle, \langle C \rangle, \langle \boldsymbol{b} \rangle)$.*

Proof. The inequalities $\psi(\boldsymbol{y})^T\boldsymbol{x} \geq -\boldsymbol{y}_d$ we add in Line 12 only depend on the vector $\boldsymbol{y}$ which is $\boldsymbol{y} = C\boldsymbol{x}^*$ and $\boldsymbol{x}^*$ is an optimal extreme point solution to the LP $P_1(\lambda(\boldsymbol{v}))$: $\min\{\lambda(\boldsymbol{v})C\boldsymbol{x} \mid A\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \in \mathbb{R}^n\}$. An optimal extreme point solution to $P_1(\lambda(\boldsymbol{v}))$ is a unique solution to the linear system

$$\begin{aligned} \boldsymbol{a}_i^T\boldsymbol{x} &= \boldsymbol{b}_i, i \in I_P(\boldsymbol{x}^*) \\ \boldsymbol{x} &\in \mathbb{R}^n \end{aligned}$$

Thus, the encoding length of $\boldsymbol{x}^*$ is bounded by $\text{poly}(\langle A \rangle, \langle \boldsymbol{b} \rangle)$. The encoding length of $\boldsymbol{y}$ then is bounded by $\text{poly}(\langle A \rangle, \langle C \rangle, \langle \boldsymbol{b} \rangle)$. □

Note though, that the encoding length of an extreme point solution of the LPs $P_1(\lambda(\boldsymbol{v}))$ is independent of the vector $\lambda(\boldsymbol{v})$; the time to solve $P_1(\lambda(\boldsymbol{v}))$ might not be. But now we can bound the running times of weakly polynomial algorithms for solving these LPs by the following lemma:

Lemma 5.21. *The encoding length of an intermediate extreme point $\boldsymbol{v}$ to solve $P_1(\lambda(\boldsymbol{v}))$ is bounded by* $\text{poly}(\langle A \rangle, \langle C \rangle, \langle \boldsymbol{b} \rangle)$.

Proof. Let j be the number of inequalities we already found until point $\boldsymbol{v}$ is discovered and let $y_1, \ldots, y_j$ be the vectors $\boldsymbol{y} = C\boldsymbol{x}$ for solutions $\boldsymbol{x}$ to P_1 for each inequality. By the previous lemma we know that $\langle \boldsymbol{y} \rangle \in \text{poly}(\langle A \rangle, \langle C \rangle, \langle \boldsymbol{b} \rangle)$. The vectors $\boldsymbol{v}$ are extreme points of L which is defined as

$$\begin{aligned} \psi(\boldsymbol{y}^i)^T\boldsymbol{x} &\geq -\boldsymbol{y}_d^i, i \in [j] \\ \boldsymbol{x} &\in \mathbb{R}^d. \end{aligned}$$

So we can again write $\boldsymbol{v}$ as the unique solution to the linear system

$$\begin{aligned} \psi(\boldsymbol{y}^i)^T \boldsymbol{x} &= -\boldsymbol{y}^i_d, i \in I_L(\boldsymbol{v}) \\ \boldsymbol{x} &\in \mathbb{R}^d. \end{aligned}$$

□

As in the investigation of the primal algorithm, let's first assume we have access to an oracle $\mathcal{A}_{P_1}$ solving problem P_1 for us. Then the running time of the dual algorithm breaks down to the following elements:

Lemma 5.22. *Let v_f be the number of $\mathcal{K}$-maximal faces of $\mathcal{D}$. Then the Dual Benson Algorithm*

1. *runs oracle $\mathcal{A}_{P_1}$ at most $\mathcal{O}(v_f)$ times,*
2. *performs a vertex enumeration at most v_f times with a running time of $\mathcal{O}({v_f}^{\lfloor \frac{d}{2} \rfloor} + v_f \log v_f)$,*
3. *has an additional overhead of $\mathcal{O}({v_f}^{\lfloor \frac{d}{2} \rfloor} \log v_f)$ for every vertex enumeration, and*
4. *removes redundant extreme points in the end with a running time of $\text{poly}(d, N, n)$.*

when fixing the number of objectives $d \geq 2$.

Proof.
1. We solve P_1 in each iteration and since we either find an extreme point or a face supporting inequality in a $\mathcal{K}$-maximal face in each iteration we have at most $\mathcal{O}(v_f)$ iterations.
2. In each iteration in which we do not find an extreme point, we also conduct a vertex enumeration, i.e., v_f times.
3. Additionally, we have to identify the extreme points of L which have not yet been checked. To find a point $v \in M \backslash E$, we can sort both sets and find the first index were the sequences differ or maintain E in a sorted manner, which does not change the asymptotic running time. By the upper bound theorem we know that $|M| \in \mathcal{O}({v_f}^{\lfloor \frac{d}{2} \rfloor})$, as the number of inequalities which define L is at most the number of faces of $\mathcal{D}$. Thus sorting takes time $\mathcal{O}({v_f}^{\lfloor \frac{d}{2} \rfloor} \log v_f)$ and we do this every time we perform a vertex enumeration.
4. This reduces to a redundancy removal problem which can be solved in time $\text{poly}(d, N, n)$ [80]. The exact running time is of no further interest here.

□

By using that the the number of $\mathcal{K}$-maximal faces of $\mathcal{D}$ is the number of faces of $\mathcal{P}$ and thus can be bounded by $\mathcal{O}(N^{\lfloor \frac{d}{2} \rfloor})$ for N being the number of extreme points or facets of $\mathcal{P}$, we arrive at the following theorem:

Theorem 5.23. *Let v_e be the number of extreme points of $\mathcal{P}$ and $d \geq 2$ be fixed. Then, Algorithm 2 has a running time bounded by*

$$\mathcal{O}(v_e{}^{\lfloor \frac{d}{2} \rfloor} \operatorname{poly}(\langle A \rangle, \langle C \rangle, \langle \boldsymbol{b} \rangle) + v_e{}^{\frac{d^2}{4}+\lfloor \frac{d}{2} \rfloor} \log v_e + \operatorname{poly}(d, v_e, n)).$$

Note though that by solving $\text{MOLP}^{\text{YF}_\text{A}}$, we do not need to remove redundant extreme points of $\mathcal{P}$ and thus can arrive at a better bound:

Corollary 5.24. *Let v_F be the number of facets of $\mathcal{P}$ and $d \geq 2$ be fixed. Then, Algorithm 2 has a running time bounded by*

$$\mathcal{O}(v_F{}^{\lfloor \frac{d}{2} \rfloor} \operatorname{poly}(\langle A \rangle, \langle C \rangle, \langle \boldsymbol{b} \rangle) + v_F{}^{\frac{d^2}{4}+\lfloor \frac{d}{2} \rfloor} \log v_F).$$

Observe that both running times are worse than those we achieved with the primal algorithm. But as in the primal algorithm, we can improve the dual algorithm by finding facet supporting inequalities exclusively and this is the topic of the next section.

5.3.2. Lexicographic Dual Benson: Finding Extreme Points Exclusively

The goal of this section is to restrict the algorithm to find facet supporting inequalities exclusively. This can be achieved by looking closer at the MOLP duality theory by F. Heyde and A. Löhne [75]. They prove a correspondence between extreme points of $\mathcal{P}$ and facets of $\mathcal{D}$.

Lemma 5.25 (Corollary 2 by Heyde and Löhne [75])**.** *$\boldsymbol{y} \in \operatorname{vert} \mathcal{P} \iff \{\boldsymbol{x} \in \mathbb{R}^d \mid \psi(\boldsymbol{y})^T \boldsymbol{x} = -\boldsymbol{y}_d\}$ supports $\mathcal{D}$ in a facet.*

We observe that computing an optimal solution $\boldsymbol{x}$ of P_1 in Line 7 of Listing 2 with lexicographic minimal $\boldsymbol{y} := C\boldsymbol{x}$ gives us a vertex $\boldsymbol{y}$ of $\mathcal{P}$ by using Lemma 5.9. Consequently, $\{\boldsymbol{x} \in \mathbb{R}^d \mid \psi(\boldsymbol{y})^T \boldsymbol{v} = -\boldsymbol{y}_d\}$ supports $\mathcal{D}$ in a facet. Moreover, the second component of the pairs we add in Line 14 are already vertices of $\mathcal{P}$ then and we can skip Line 15 and do not need to remove redundant pairs anymore.

We define lex-$P_1(\boldsymbol{\ell})$: $\operatorname{lexmin}\{(\boldsymbol{\ell}^T C\boldsymbol{x}, \boldsymbol{c}_1\boldsymbol{x}, \ldots, \boldsymbol{c}_d\boldsymbol{x}) \mid A\boldsymbol{x} \geq \boldsymbol{b}, \boldsymbol{x} \in \mathbb{R}^n\}$, where $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_d$ are the rows of C and change Lines 1 and 7 accordingly. Because of Lemma 5.16, we can solve these lexicographic linear optimization problems in polynomial time. Thus again, we do not lose anything by converting this problem to a lexicographic LP.

So let us again break the running time of the algorithm down into its parts:

Lemma 5.26. *Let v_e (v_F) be the number of extreme points ($\mathcal{K}$-maximal facets) of $\mathcal{D}$. Then the Dual Benson Algorithm*

1. *runs oracle $\mathcal{A}_{P_1}$ at most $v_e + v_F$ times,*
2. *performs a vertex enumeration at most v_F times with a running time of $\mathcal{O}(v_F{}^{\lfloor \frac{d}{2} \rfloor} + v_F \log v_F)$, and*

3. has an additional overhead of $\mathcal{O}(v_F{}^{\lfloor \frac{d}{2} \rfloor} \log v_F)$ *for every vertex enumeration.*

when fixing the number of objectives $d \geq 2$.

By using that the number of extreme points of $\mathcal{D}$ is the number of faces of $\mathcal{P}$ and the number of $\mathcal{K}$-maximal faces of $\mathcal{D}$ is the number of extreme points of $\mathcal{P}$, we arrive at the running times for the algorithm:

Theorem 5.27. *Let* v_e *(*v_F*) be the number of extreme points (facets) of* $\mathcal{P}$*. Then the improved version of the dual of Benson's algorithm has a running time of*

$$\mathcal{O}((v_e + v_F)\,\mathrm{poly}(\langle A\rangle, \langle C\rangle, \langle \boldsymbol{b}\rangle) + v_e{}^{\lfloor \frac{d}{2} \rfloor + 1} \log v_e)$$

for each fixed $d \geq 2$*. Moreover, problem* $MOLP^{\mathrm{YEx}}$ *can be solved in time*

$$\mathcal{O}(v_e{}^{\lfloor \frac{d}{2} \rfloor}\,\mathrm{poly}(\langle A\rangle, \langle C\rangle, \langle \boldsymbol{b}\rangle) + v_e{}^{\lfloor \frac{d}{2} \rfloor + 1} \log v_e)$$

for each fixed $d \geq 2$ *and* $MOLP^{\mathrm{YFa}}$ *can be solved in time*

$$\mathcal{O}(v_F{}^{\lfloor \frac{d}{2} \rfloor}\,\mathrm{poly}(\langle A\rangle, \langle C\rangle, \langle \boldsymbol{b}\rangle) + v_F{}^{\frac{d^2}{4} + \lfloor \frac{d}{2} \rfloor} \log v_F)$$

for each fixed $d \geq 2$ *if* $y_{\min}$ *exists for all these problems.*

This is a significant improvement over Theorem 5.23, since we eliminate the term having d^2 in the exponent. Moreover, we are now able to bound the delay of the algorithm. In the original algorithm this was not possible since it could take a large number of iterations until the $(d+1)$-th extreme point is found.

Theorem 5.28. *Let* $d \geq 2$ *be fixed. For the lexicographic dual Benson algorithm the* k*-th delay is in*

$$\mathcal{O}(k^{\lfloor \frac{d}{2} \rfloor}(\mathrm{poly}(\langle A\rangle, \langle C\rangle, \langle \boldsymbol{b}\rangle) + \log k))$$

for solving $MOLP^{\mathrm{YEx}}$ *with existing* $\boldsymbol{y}_{\min}$.

Proof. The first delay only consists of solving the problem lex-$P_1(\boldsymbol{e}_1)$ and can thus be bounded by $\mathcal{O}(\mathrm{poly}(\langle A\rangle, \langle C\rangle, \langle \boldsymbol{b}\rangle))$. The k-th delay is the time between outputting the k-th and the $(k+1)$-th extreme point. We have at most as many iterations as the intermediate polyhedron has extreme points. Since the intermediate polyhedron has $k+d$ inequalities, it can have at most $\mathcal{O}((k+d)^{\lfloor \frac{d}{2} \rfloor})$ extreme points. Thus, the number of iterations is bounded by $\mathcal{O}(k^{\lfloor \frac{d}{2} \rfloor})$ for fixed d. In the v_e-th delay, we explore at most all extreme points of $\mathcal{D}$ and do not enter the else-branch at all. □

5.4. Application to Multiobjective Combinatorial Optimization

Let us now get back to where we started: The motivation for considering MOLP was to find extreme points of MOCO problems. In the introduction to this chapter we

showed, that we can try solving the convex relaxation Definition 5.1 to find extreme points of a given MOLP.

One concern is that the algorithms only work if the ideal point $\boldsymbol{y}_{\min}$ exists. But this is always the case for feasible combinatorial instances, because there is only a finite number of solutions. If we now have an algorithm which gives us for every instance of a MOCO problem a corresponding MOLP instance, we can compute the extreme points in an output-sensitive way. Let us formalize the notion of having such an algorithm: Let (I, S, v) be a MOCO problem. We call an algorithm, which computes for every instance $x \in I$ a matrix $A \in \mathbb{Q}^{m \times n}$, a matrix $C \in \mathbb{Q}^{d \times n}$ and a vector $\boldsymbol{b} \in \mathbb{Q}^m$ in polynomial time in $|x|$, a *compact formulation* if $\operatorname{conv} S(x) = \{\boldsymbol{x} \in \mathbb{R}^n \mid A\boldsymbol{x} \leq \boldsymbol{b}\}$ and $C = C_x$. Note that the polynomial-time requirement implies that $\langle A\rangle, \langle C\rangle, \langle b\rangle \in \operatorname{poly}(|x|)$.

Additionally, in Lemma 5.16 we showed that the lexicographic LP can then also be solved in polynomial time, thus we can use the improved version of the dual of Benson's algorithm and arrive at the following corollary:

Corollary 5.29. *If a compact formulation exists for a MOCO problem O, then O^{YEx} can be solved in incremental polynomial time for every fixed $d \geq 2$.*

This is a restriction, because there are combinatorial polytopes, even with polynomial-time solvable optimization problems, which do not have compact formulations as for example the matching polytope (cf. Rothvoss [117]). But we can try to generalize the above corollary to more problems: Revisiting the (unimproved) Dual Benson algorithm, we see that the only point where the algorithm reads the input is in solving the problems $P_1(\boldsymbol{\ell})$ with varying $\boldsymbol{\ell}$. Using the convex relaxation, we need to solve the LP: $\min\{\boldsymbol{\ell}^T C_x \boldsymbol{x} \mid \boldsymbol{x} \in \operatorname{conv} S(x)\}$. We can observe that we are only interested in extreme point solutions to this LP and in this case we can solve the weighted-sum problem $\min\{\boldsymbol{\ell}^T C_x \boldsymbol{x} \mid \boldsymbol{x} \in S(x)\}$ instead. We do not have to worry about the encoding of the objective function vector $\boldsymbol{\ell}^T C$, since Lemma 5.21 guarantees that the encoding length is bounded by a polynomial in the input size.

So we can generalize Corollary 5.29 to all problems for which the weighted-sum problem can be solved in polynomial time.

Theorem 5.30. *If for a MOCO problem O, P_1 can be solved in polynomial time, O^{YEx} can be solved in output-polynomial time for every fixed $d \geq 2$.*

On the flip side though, we cannot use the improved version when we only have access to an algorithm solving the weighted-sum problem; so we can only prove output-polynomiality. But if we have also access to an algorithm solving the lexicographic version of the weighted sum problem, we can use the dual algorithm and arrive at incremental polynomial time

Theorem 5.31. *If for a MOCO problem O, lex-P_1 can be solved in polynomial time, O^{YEx} can be solved in incremental polynomial time for every fixed $d \geq 2$.*

If we apply Theorem 5.31 to the multiobjective spanning tree problem, we immediately obtain an algorithm with a polynomial upper bound on the running time with

respect only to the input size to find the nondominated extreme points for each fixed number of objectives. This is well known in the biobjective (or parametric) case, but it is a new result for the general case. It bases on the well known fact that for the MOST problem there exist at most $\mathcal{O}(m^{2(d-1)})$ nondominated extreme points as shown by Ganley, Golin, and Salowe [68]. We obtain a similar result for the multiobjective global min-cut problem, as it has been shown by Armon and Zwick [7] that the parametric complexity is polynomial in the input size for any fixed number of objectives. For the MOSP a polynomial parametric complexity was proven by Nikolova et al. [103], consequently we also obtain a polynomial running time for $MOSP^{\mathrm{YEx}}$. We summarize these findings in a corollary:

Corollary 5.32. *The problems $MOSP^{\mathrm{YEx}}$, $MOST^{\mathrm{YEx}}$ and $MOMC^{\mathrm{YEx}}$ can be solved in polynomial time for every fixed number of objectives.*

5.4.1. Computational Study

In the remainder of this chapter we want to change our focus from theoretical considerations to more practical ones. In the introduction we said that we want to support algorithm designers in forging more efficient algorithms. In this section we show that through having a running time analysis framework, we are able to select efficient algorithms for a specific task.

The problem of solving O^{YEx} for a MOCO problem O receives its importance through the need for algorithms for the first phase of the two phases method. There do exist algorithms and implementations which solve this problem, most recently by A. Przybylski, X. Gandibleux and M. Ehrgott [114] and Ö. Özpeynirci and M. Köksalan [106]. We call the first one the PGE-algorithm and the latter one the OK-algorithm. While a running time analysis is not conducted in either work, the authors show empirical results of their implementations. In the PGE case, a special implementation for three objectives is showcased, while an implementation of the algorithm for four and more objectives is described but not implemented. The authors state that the algorithm for four and more objectives is most certainly not practically efficient. For the OK-algorithm, Özpeynirci and Köksalan have an implementation for the general case of arbitrary many objectives and show empirical results for the case of three and four objectives only, because the implementation was not able to solve larger instances. Implementations for either algorithm were not available.

We decided to compare our algorithm to the OK algorithm, because it is capable of solving an arbitrary number of objectives. The PGE implementation is a specialized implementation for the three objective case and thus can be much faster than the Dual Benson implementation on these instances. Additionally, comparing the running times stated by Przybylski, Gandibleux, and Ehrgott [114] to our empirical results, the PGE implementation is quite efficient for the three objective case and is always a slight bit faster than our implementation.

To compare the implementations, we used multiobjective assignment instances from the literature as well as newly generated ones, because the available instances are too small to highlight the capabilities of the Dual Benson approach.

The Lexicographic Dual Benson Implementation

The Lexicographic Dual Benson algorithm was implemented in C++ using double precision arithmetic and compiled using LLVM 3.3. We use the formulation which is described in Listing 2 as a base line. To mitigate numerical inconsistencies, we do not touch the weights $\lambda(\boldsymbol{v})$ as is done in the work by Ehrgott, Löhne, and Shao [59], where the authors optimize along a line through $\boldsymbol{v}$ and a point in the interior of $\mathcal{D}$, possibly introducing longer encodings of the numbers in the input of these LPs. We use `std::vector` where possible. To hold vector information, we also use `std::vector` which does introduce a possibly avoidable additional indirection, but it makes the source code very clear and easily usable for any number of objectives without recompilation. To compute the extreme points of the intermediate polyhedra, we implemented a version of the Double Description method with full adjacency information in the case of $d \in \{3, 4\}$ and for $d > 4$, we used the CDD library (cf. Fukuda and Prodon [65]). To find lexicographic minimal assignments for the lexicographic dual Benson algorithm, we implemented a lexicographic version of the Hungarian method.

The OK Implementation

One very important caveat must be stated here: The PGE algorithm as well as the OK algorithm were not able to find all extreme points of all given instances and sometimes found points which are not extreme. This is due to numerical inconsistencies occurring through the use of double precision arithmetic.

This is especially a problem in the original implementation of the OK algorithm, which is not easy to implement efficiently and correct. Let us describe the algorithm in some detail. The algorithm proceeds in iterations and a set of already computed extreme points R is maintained. In each iteration the algorithm picks a subset of R of cardinality d: $Y = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_d\}$ and computes a normal $\boldsymbol{\ell}$ to the hyperplane affinely spanned by Y by solving the linear system

$$\begin{aligned} \boldsymbol{y}_i^T \boldsymbol{x} &= 0, i \in [d] \\ \mathbf{1}^T \boldsymbol{x} &= 1 \\ \boldsymbol{x} &\in \mathbb{R}^d. \end{aligned}$$

If all points in Y happen to lie in a common nontrivial face of $\operatorname{conv} R$ then $\boldsymbol{\ell}$ is a normal to this face. In this case the weighted-sum problem $\min\{\boldsymbol{\ell}^T C\boldsymbol{x} \mid \boldsymbol{x} \in S(x)\}$ gives us a new point on the boundary of $\mathcal{P}$ iff the face is not a face of $\mathcal{P}$. If the points in Y do not have a nontrivial face in common, solving the weighted sum problem gives us a point we already found. Since the algorithm does not know the actual face structure of $\operatorname{conv} R$, this happens on a regular base. Özpeynirci and Köksalan describe pruning rules to mitigate this problem and compute weighted-sum problems of not promising sets Y less often.

The severe numerical inconsistencies in the implementation are introduced in the initialization of the algorithm: The first set of extreme points are artificially introduced

"dummy" points $M\boldsymbol{e}_i$ with $M \in \mathbb{Q}$ and $i \in [d]$. The hyperplane in the affine span of all these $M\boldsymbol{e}_i$ must not separate any point of the nondominated set from the origin, or this point might possibly not be found (assuming minimization and $\boldsymbol{y} \geq \mathbf{0}$ for every point $\boldsymbol{y}$ of $\mathcal{Y}_N$). As a solution to this problem, Özpeynirci and Köksalan propose to set M to be "large enough". A lower bound was shown in the dissertation by Ö. Özpeynirci from 2008 [105] to be $(\sum_{i\in[d]} \max_{\boldsymbol{y}\in\mathcal{Y}_X}\{\boldsymbol{y}_i\}) \cdot \max_{i\in[d]} \max_{\boldsymbol{y}\in\mathcal{Y}_X}\{\boldsymbol{y}_i\}$. This results in linear systems containing numbers of very different precision, which in turn results in numerical inconsistencies when using double precision arithmetic such as the IEEE 754 standard.

Instead of using points with large encoding length on the axes, we use projective points at infinity to reduce the numerical inconsistencies which occurred in the experimental setting by Özpeynirci and Köksalan [106]. Nevertheless, the implementation still misses some nondominated extreme points, but does always find more than the original implementation. To compute optimal assignment solutions in the OK algorithm, we use an implementation of the Hungarian method.

We also implemented the pruning steps discussed by Özpeynirci and Köksalan [106] and as a sanity check we compared the running times of our implementation to the running times stated in the work by Özpeynirci and Köksalan. Our implementation of the OK algorithm is always much faster than the running times stated in the paper, but this is only a rough estimate, since the paper was submitted in 2008 and only a notebook was used to perform the experiments.

Questions for Empirical Investigation

We ask two questions: The first one being which algorithm performs better on a set of benchmark instances (Q1). Our metric is the running time, since both algorithms are exact algorithms.

The second question is asked in connection to the actual improvement we proposed in Section 5.3.2: Does this improvement also give us an improvement in the practical running time on the benchmark instances? (Q2)

Instances Used

Both implementations were tested on instances of the multiobjective assignment (or minimum weight bipartite matching) (MOAP) problem. The MOAP problem is often used as a benchmark problem in the biobjective case, but is also used in the computational studies of the approaches existing for three and four objectives (cf. Özpeynirci and Köksalan [106] and Przybylski, Gandibleux, and Ehrgott [114]). The instances consist of complete bipartite graphs with $2n$ nodes, where n is said to be the number of resources. The weights in the instances from Özpeynirci and Köksalan [106] are chosen uniformly at random from the set $\{1, \ldots, 20\}$. We note that on these instances the nondominated set does not grow exponentially in n, because the numbers in the input are bounded by a constant.

The instances available from Özpeynirci and Köksalan [106] have three or four

objectives and for every number of objectives and resource combination there are 20 instances. The instances with three objectives have 10, 20 or 30 resources, and there is only one instance set with 10 resources and four objectives. We extended the set of instances and generated similar instances with up to six objectives[1].

Experimental Setting

The experiments were performed on an Intel Core i7-3770, 3.4 GHz and 16 GB of memory running Ubuntu Linux 12.04. To bound the running times within reason, we enforce a running time limit of one hour. Additionally, because of the hardware resource used, we enforce a memory limit of 16 GB.

Evaluation Metrics

To evaluate the results, we use the median running time and median number of extreme points. It is easy to see that the distribution of running times is heavily skewed, since negative running times do not exist. When using the median as a measure, the standard deviation is not an appropriate measure of deviation. This is why we use the median absolute deviation (MAD) for this, which is a robust measure of dispersion. For more information on the MAD, see, e.g., Hoaglin, Mosteller, and Turkey [76] and Leys et al. [95].

5.4.2. The Results

In this section, we present the results and answer our two questions by means of empirical data.

Q1: Which algorithm performs better?

In Table 5.1 we can see all results showing the running times and numbers of extreme points found. We observe that the Lexicographic Dual Benson implementation is able to solve all instances in the given limits. The implementation of the OK algorithm is only able to solve very small instances with three and four objectives. In all other cases there are instance classes on which the implementation cannot solve all of the instances, prohibiting a statistical analysis. Nevertheless, we give the median and MAD in parentheses if the OK implementation was not able to solve all instances. In most cases, memory was the limiting factor for the OK implementation. This is not surprising as many tuples of extreme points survive the pruning steps. In the cases where the OK implementation is able to solve all instances, we see that the lexicographic dual Benson implementation is up to a factor of 640 times faster.

Without further statistical analysis we can clearly say that the Lexicographic Dual Benson implementation is much more usable in practice, at least for MOAP instances.

[1] All instances are available at `https://ls11-www.cs.tu-dortmund.de/staff/boekler/moco-instances`

		Lexicographic Dual Benson				**OK implementation**				
		Running Time [s]		$\lvert\mathcal{Y}_X\rvert$		Running Time [s]		$\lvert\mathcal{Y}_X\rvert$		no. solved
d	n	Median	MAD	Median	MAD	Median	MAD	Median	MAD	#
3	10*	**0.01**	0.002	31.5	4.448	**0.03**	0.010	31.5	4.448	20
	20*	**0.11**	0.014	150.5	17.050	6.31	1.491	150.5	17.050	20
	30*	**0.70**	0.120	368.5	51.150	160.24	57.858	368.5	51.150	20
	40	**2.57**	0.220	709	51.150	1660.63	542.646	709.0	51.150	20
	50	**6.77**	0.617	1015.5	111.195	—	—	—	—	0
	60	**18.47**	1.434	1603.5	146.036	—	—	—	—	0
	70	**35.31**	1.770	2109.5	108.230	—	—	—	—	0
	80	**67.22**	4.619	2819	185.325	—	—	—	—	0
	90	**120.60**	8.107	3523	272.057	—	—	—	—	0
	100	**203.77**	4.153	4403	205.340	—	—	—	—	0
	110	**312.59**	25.255	5309	395.113	—	—	—	—	0
	120	**472.45**	34.327	6192.5	385.476	—	—	—	—	0
	130	**701.87**	52.801	7446.5	471.467	—	—	—	—	0
	140	**961.27**	32.992	8362	272.057	—	—	—	—	0
	150	**1349.74**	87.874	9626	374.357	—	—	—	—	0
	160	**1840.59**	106.013	10881.5	320.242	—	—	—	—	0
	170	**2532.50**	81.713	12282.5	556.716	—	—	—	—	0
4	10*	**0.06**	0.019	102.5	26.687	3.29	2.447	102.5	26.687	20
	15	**0.33**	0.079	453.5	97.852	(253.35)	(105.681)	(347.0)	(14.826)	7
	30	**12.97**	1.824	3646.0	444.039	—	—	—	—	0
	50	**270.74**	34.613	17334.0	1200.906	—	—	—	—	0
	60	**789.04**	65.129	30476.5	1611.586	—	—	—	—	0
	70	**1978.68**	269.907	48667.0	5777.692	—	—	—	—	0
5	8	**0.22**	0.131	125.5	34.100	(29.91)	(36.378)	(124.0)	(31.876)	18
	14	**33.30**	16.416	1228.0	313.570	—	—	—	—	0
	20	**1055.82**	252.966	5052.5	699.787	—	—	—	—	0
6	6	**0.16**	0.094	75.0	20.015	(39.01)	(50.977)	(73.0)	(17.791)	18
	8	**1.73**	1.096	228.0	54.856	(159.25)	(45.658)	(164.5)	(20.015)	2
	10	**25.62**	14.225	703	153.449	—	—	—	—	0
	12	**213.95**	159.990	1798.0	549.303	—	—	—	—	0

Table 5.1.: Computational results on multiobjective assignment instances with d objectives and n resources. Instances with an * were taken from the work by Özpeynirci and Köksalan [106].

	Running Times $\mathcal{T}_{\text{lex}}/\mathcal{T}$						**Points found**	
	Hungarian algorithm		VE		Total		**lex / no lex**	
n	Median	MAD	Median	MAD	Median	MAD	Mean	σ
20	1.114	0.0301	1.000	0.0091	1.002	0.0086	1.000	0.0003
22	1.118	0.0221	1.000	0.0046	1.000	0.0045	1.000	0.0002
24	1.130	0.0110	0.999	0.0055	1.000	0.0055	1.000	<0.0001
26	1.126	0.0170	0.999	0.0044	0.999	0.0045	1.000	<0.0001

Table 5.2.: Comparison of the dual Benson implementation with and without lexicographic oracle on multiobjective assignment instances with $d = 5$.

Q2: Does the improvement of the Dual Benson algorithm give us benefit in practice?

In the second set of experiments, we compare the practical performance of the dual Benson algorithm when using our theoretical improvements from Section 5.3.2 to the original variant. The same instances as in the previous experiments were used. In Table 5.2 we present the experiments with five objectives. The table displays the medians and MADs of the quotients of the lexicographic variant over the non-lexicographic variant. The table shows these statistics for the cumulated running time of the Hungarian algorithm, the vertex enumeration (VE) and the total time. In addition, the last column of Table 5.2 displays the mean and standard deviation of the quotient of the number of points found by both algorithms. Median and MAD are 1 and 0, respectively, for every entry of the last column.

We observe that the running times are very similar. The quotients of the total running time medians are very close to 1. On one hand, the vertex enumeration is only slightly faster when using a lexicographic oracle. On the other hand, the cumulated time of the lexicographic oracle is always slower than the time of the original Hungarian method. Of course, the vertex enumeration dominates the total running time, but we also observe that it does not happen too often that redundant inequalities are found.

We can also observe that the medians of the total running time quotients seem to shrink when increasing the number of resources. In order to observe if this trend continues, we need to test much larger instances which is not possible with the current implementation, because of running times of already more than 12 hours on instances with 26 resources.

The experiments show that on small instances the difference is small and in favor of the unimproved variant of the Dual Benson implementation. It would be very interesting to test this on instances of sizes up to 50 or 70 resources, but this is on the other hand not possible in the current implementation.

6. Showing Hardness

In the previous chapter, we demonstrated how we can show that a problem is an easy problem in the sense of output-sensitive complexity. We now have first examples of easy problems: the multiobjective (global) min-cut problem (cf. Section 4.4) and the multiobjective linear programming problem as well as enumerating extreme nondominated points. In this chapter, we show how output-sensitive complexity can be used to prove that multiobjective optimization problems cannot be solved efficiently under mild complexity theoretic assumptions. We approach hardness from two different perspectives:

Our first example is the multiobjective shortest path problem, or MOSP. This very well known MOCO problem admits fairly good algorithms in practice, at least for a small number of objective functions. We see examples for this in Part II. Nevertheless, computational running time guarantees beyond classical intractability are not known. So the natural question arises if there exists an output-sensitive algorithm for MOSP. As we see in Section 6.1, this is most probably not the case: We prove that the existence of an output-sensitive algorithm for MOSP implies $\mathbf{P} = \mathbf{NP}$.

On the other hand, there are problems for which such a result cannot be easily found. Especially the biobjective unconstrained optimization (BUCO) problem is an example of a very easy looking problem, to which we are unable to assign a label "easy" or "hard". But we can show that if there is no output-sensitive algorithm for BUCO then there is none for many other multiobjective optimization problems, giving rise for a complexity class based on the BUCO problem which we introduce in Section 6.2.

6.1. Multiobjective Shortest Path

In this section, we give an example of a problem, which is not solvable in an output-sensitive way if $\mathbf{P} \neq \mathbf{NP}$. The problem we consider is the multiobjective shortest path problem (MOSP) as in Definition 2.19. In practice, there are many examples where the MOSP problem can be solved relatively well. But usually algorithms which solve MOSP solve a more general problem which we call the *multiobjective single-source shortest-path (MO-SSSP) problem*. In the MO-SSSP problem, instead of enumerating one Pareto-front of the Pareto-optimal s-t-paths, we enumerate for every vertex $v \in V \backslash \{s\}$ the Pareto-front of all Pareto-optimal s-v-paths. One example of such an algorithm is the well known label setting algorithm by Martins [98]: It is easy to show that it solves the MO-SSSP problem in incremental polynomial time.

We stress here that MOSP and MO-SSSP are indeed different problems. The insight that the MO-SSSP problem is solvable in incremental polynomial time does not immediately transfer to the MOSP problem. This can be seen, when using Martins'

algorithm for solving MOSP on a given instance, where we cannot even guarantee output-polynomial running time.

More modern label setting and label correcting algorithms usually start from there and try to avoid enumerating too many unneccessary s-v-paths. But how many unnecessary s-v-paths—or states—can be pruned? Can we construct an algorithm that does not compute too many, i.e., at most polynomially more labels than necessary? The following considerations show that a large number of these s-v-paths are needed in these algorithms and that—in contrast to the MO-SSSP problem—there is no output-sensitive algorithm solving MOSP in general, if we assume $\mathbf{P} \neq \mathbf{NP}$.

Moreover, we exemplify a general methodology in output-sensitive complexity for showing that there is no output-sensitive algorithm unless $\mathbf{P} = \mathbf{NP}$. Similar to single objective optimization, we show that an enumeration problem is hard by showing the hardness of a decision problem, which is defined as follows.

Definition 6.1 (Finished Decision Problem, Schmidt [121])**.** Given an enumeration problem $E = (I, C)$, E^{FIN} is the following language:

$$\{(x, M) \in I \times C(I) \mid x \in I \text{ and } M = C(x)\}.$$

That is, when we are given an instance $x \in I$ of the enumeration problem (I, C) and a subset $M \subseteq C(x)$ of the configuration set, we ask the question: Do we already have all configurations?

Lawler, Lenstra, and Rinnooy Kan [93] show for the example of an independence system, that if there is an output-polynomial algorithm for the problem of enumerating all maximal independent sets, then we can solve the associated finished decision problem in polynomial time. In his Diploma Thesis, J. Schmidt [121] generalizes this to general enumeration problems, but with the additional requirement that $C(x)$ for a given instance $x \in I$ needs to be polynomial time decidable. We now state a very similar proof which gets rid of the necessity of $C(x)$ being decidable in polynomial time. This is important, because for most MOCO problem, deciding if a given point in $\mathbb{Q}^d$ is a nondominated point, is either unknown to be polynomial time decidable or **NP**-hard.

Lemma 6.1. *If $E \in$ **TotalP** then $E^{\text{FIN}} \in$ **P**.*

Proof. Let $E = (I, C)$ be given. Since $E \in$ **TotalP**, there exists a polynomial function p and a RAM A which enumerates $C(x)$ for a given $x \in I$ in time at most $p(|x|, |C(x)|)$.

We now construct an algorithm which decides for a given instance (x, M) of E^{FIN} whether it is a "Yes"- or "No"-instance. We simulate A on x for time $p(|x|, |M|)$. If A does not halt on x, then $|M| < |C(x)|$ and we can safely answer "No". If A does halt on x, we are almost done. It can still be the case that the input M was invalid, i.e., $M \backslash C(x) \neq \emptyset$ or that $M \subsetneq C(x)$ and the algorithm was by chance faster than expected. Both conditions can be tested by checking whether M is equal to the output of A. If it is, we return "Yes", otherwise "No".

Simulation takes time $\text{poly}(|x|, |M|)$. The output of A has at most size $\text{poly}(|x|, |M|)$ and the final check can thus be done in $\text{poly}(|x|, |M|)$. □

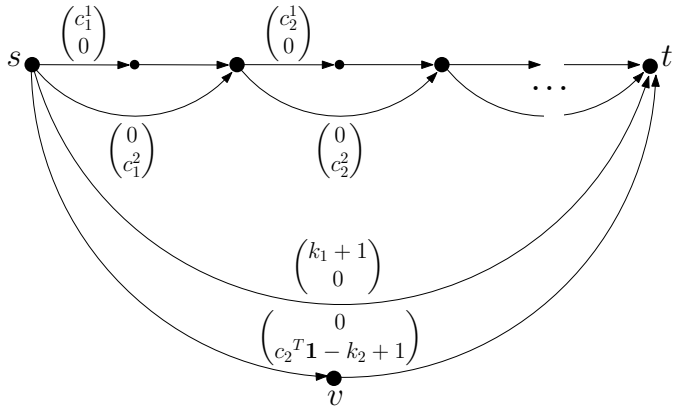


Figure 6.1.: Showing the reduction in the proof for Theorem 6.2. Arcs with no label have cost $\mathbf{0}$.

We mainly use the contraposition of this lemma: If the finished decision problem cannot be solved in polynomial time, then we cannot solve the enumeration problem in output-polynomial time. We can now use this method to finally show that the MOSP problem is indeed a hard enumeration problem.

Theorem 6.2. *There is no output-sensitive algorithm for the MO s-t-path problem unless* $\boldsymbol{P} = \boldsymbol{NP}$*, even if* $d = 2$ *and the input graph is outerplanar.*

Proof. We show that MOSP$^{\text{FIN}}$ is **co-NP**-hard. By Lemma 6.1 this shows that a **TotalP** algorithm for MOSP implies $\mathbf{P} = \mathbf{co\text{-}NP}$ and thus $\mathbf{P} = \mathbf{NP}$.

We do this by reducing instances of the complement of the Knapsack problem:

$$\text{(KP) } \{(\boldsymbol{c}^1, \boldsymbol{c}^2, k_1, k_2) \mid \boldsymbol{c}^{1T}\boldsymbol{x} \leq k_1, \boldsymbol{c}^{2T}\boldsymbol{x} \geq k_2, \boldsymbol{x} \in \{0,1\}^n\}\,.$$

Without loss of generality, we can assume that $\boldsymbol{c}^1, \boldsymbol{c}^2 \in \mathbb{N}^n$, $k_1, k_2 \in \mathbb{N}$, $\boldsymbol{c}^{1T}\mathbf{1} > k_1$ and $\boldsymbol{c}^{2T}\mathbf{1} > k_2$. The problem above is still **NP**-complete under these restrictions, cf. Kellerer, Pferschy, and Pisinger [83]. And thus the complement co-KP is **co-NP**-hard.

We now construct an instance $\hat{I}$ of the $MOSP^{\text{FIN}}$ problem from an instance I of the KP problem. The instance has nodes $\{v_i^1, v_i^2\}$ for every variable $\boldsymbol{x}_i$ and one additional node v_{n+1}^1. It has one arc for every $i \in [n]$: (v_i^1, v_i^2) with cost $(\boldsymbol{c}_i^1, 0)^T$ and for every $i \in [n]$ it has one arc (v_i^1, v_{i+1}^1) with cost $(0, \boldsymbol{c}_i^2)^T$ and one arc (v_i^2, v_{i+1}^1) with cost $\mathbf{0}$. The node v_1^1 is identified with s, the node v_{n+1}^1 is identified with t. There is one additional arc (s,t) with cost $(k_1+1, 0)^T$ and one additional path (s,v,t) with cost $(0, \boldsymbol{c}^{2T}\mathbf{1} - k_2 + 1)^T$. To complete the reduction, we set $M \coloneqq \{(k_1+1, 0)^T, (0, \boldsymbol{c}^{2T}\mathbf{1} - k_2 + 1)^T\}$. An example of this reduction can be seen in Figure 6.1.

We observe that the instance is valid, and that there are at least two Pareto-optimal paths, namely (s,t) and (s,v,t) having cost $(k_1+1, 0)^T$ and $(0, \boldsymbol{c}^{2T}\mathbf{1} - k_2 + 1)^T$,

respectively. Both paths are Pareto-optimal because $\boldsymbol{c}_i^1, \boldsymbol{c}_i^2 > 0$ for all $i \in [n]$ and thus all other paths have either non-zero components in their objective function values or the sum of all values in one objective function and 0 in the other. All steps can be performed in polynomial time in the input instance I.

Now, we take an instance $I \notin$ co-KP, or equivalently $I \in$ KP. Accordingly, there exists $\boldsymbol{x} \in \{0,1\}^n$ with ${\boldsymbol{c}^1}^T\boldsymbol{x} \leq k_1$ and ${\boldsymbol{c}^2}^T\boldsymbol{x} \geq k_2 \Leftrightarrow {\boldsymbol{c}^2}^T(\mathbf{1} - \boldsymbol{x}) \leq {\boldsymbol{c}^2}^T\mathbf{1} - k_2$. Using this solution, we construct a path p in the MOSP instance $\hat{I}$: For every i with $\boldsymbol{x}_i = 1$, we take the route through node v_i^2, inducing cost of $(\boldsymbol{c}_i^1, 0)^T$. For every i with $\boldsymbol{x}_i = 0$, we take the route directly through arc (v_i^1, v_{i+1}^1), inducing cost $(0, \boldsymbol{c}_i^2)^T$. We observe that $v(\hat{I}, p)_1 = {\boldsymbol{c}^1}^T\boldsymbol{x} \leq k_1$ and $v(\hat{I}, p)_2 = {\boldsymbol{c}^2}^T(\mathbf{1} - \boldsymbol{x}) \leq {\boldsymbol{c}^2}^T\mathbf{1} - k_2$. But then, $\hat{I} \notin MOSP^{\text{FIN}}$, since p is neither dominated by (s,t) nor (s,v,t).

Now, suppose for some instance I, the constructed instance $\hat{I} \notin MOSP^{\text{FIN}}$, i.e., there is an additional nondominated path p apart from (s,t) and (s,v,t). Since it is not dominated by (s,t) and (s,v,t), it must hold that $v(\hat{I}, p)_1 \leq k_1$ and $v(\hat{I}, p)_2 \leq {\boldsymbol{c}^2}^T\mathbf{1} - k_2$. But then, we can construct a solution to KP in I as follows: The path P cannot take arcs from (s,t) or (s,v,t), so it needs to take the route through the v_i^1 and v_i^2 nodes. For every $i \in [n]$ it can only either take arc (v_i^1, v_i^2) or (v_i^1, v_{i+1}^1). If it takes the first arc, we set $\boldsymbol{x}_i \coloneqq 1$; if it takes the second arc, we set $\boldsymbol{x}_i \coloneqq 0$. This solution then has cost ${\boldsymbol{c}^1}^T\boldsymbol{x} = v(\hat{I}, p)_1 \leq k_1$ and ${\boldsymbol{c}^2}^T\boldsymbol{x} = {\boldsymbol{c}^2}^T\mathbf{1} - v(\hat{I}, p)_2 \geq {\boldsymbol{c}^2}^T\mathbf{1} - {\boldsymbol{c}^2}^T\mathbf{1} + k_2 = k_2$. Hence, $I \in$ KP or equivalently $I \notin$ co-KP.

This proves that the reduction is a polynomial-time Karp reduction from the complement of KP to the finished decision variant of MOSP and thus the theorem. □

The proof also shows that deciding whether the nondominated set of a MOSP instance is larger than 2 is **NP**-hard. Moreover, it gives us a lower bound on the quality we can approximate the size of the nondominated set.

Corollary 6.3. *It is **NP**-hard to approximate the size of the nondominated set of the MOSP problem within a factor better than $\frac{3}{2}$.*

There is one more observation we can see in this proof; it proves that the MOSP is **NP**-hard in the following way:

Proposition 6.4. *The MOSP problem is **NP**-hard (with respect to Cook-reduction).*

Proof. We prove this by reducing the decision problem whether the nondominated set is larger than 2, to the MOSP problem. Let the MOSP problem be given in the form (I, S, v). Formally the problem whether the nondominated set is larger than 2, is the following language:

$$\{x \in I \mid |\mathcal{Y}_N| > 2\}$$

We can copy the instance and give it to the oracle. The oracle returns the nondominated set and we read at most the first three entries, if they exist. If there are exactly 0, 1 or 2 entries, the algorithm returns false. Otherwise, it returns true. The reduction runs in polynomial time. □

Since MOSP can be seen as a special case of many important MOCO problems, e.g., the multiobjective minimum perfect matching problem and the multiobjective minimum-cost flow problem, these problems also turn out to not have an output-sensitive algorithm, unless $\mathbf{P} = \mathbf{NP}$. Moreover, Corollary 6.3 and Proposition 6.4 hold accordingly.

6.2. BUCO and BUCO-Hardness

We now turn to a problem we have already seen in Section 4.1: The BUCO problem. The definition can be found in Section 2.5.1. We do not know the output-sensitive complexity of this problem, but we describe some methods with which the problem can be attacked in Section 6.2.1. Since the BUCO problem can as well be a hard enumeration problem, in Section 6.2.2 we describe some approaches how to show that the BUCO problem is hard. Then in Section 6.2.3, we use that the BUCO problem is probably a hard problem and transfer this hardness to other problems by introducing a new complexity class of **BUCO**-hard problems.

6.2.1. Approaches to Solve BUCO

In Section 4.1 we describe how the Nemhauser-Ullmann algorithm [102] for the knapsack problem is capable of solving the BUCO problem in time $\mathcal{O}(nW_2)$, where $W_2 = \boldsymbol{c}^{2^T}\mathbf{1}$. Thus, a pseudo-polynomial time algorithm is available. Moreover, Ackermann et al. [1] prove that the problem can be solved in smoothed-polynomial time. The output-sensitive complexity is, however, unknown.

To try solving the BUCO problem, several approaches are possible: We could investigate the running time of the Nemhauser-Ullmann algorithm and try to find out if it is output-sensitive on BUCO instances or find a set of instances on which it has a super-polynomial running time in the input and the output. As a very natural approach, we could investigate the $\boldsymbol{\varepsilon}$-constraint method on the BUCO problem and try to deduce an output-sensitive running time there. We now discuss these possibilities in more detail.

The Nemhauser-Ullmann Algorithm

Let us start with the Nemhauser-Ullmann algorithm. For an instance defined by $\boldsymbol{c}^1, \boldsymbol{c}^2 \in \mathbb{Q}^n$, let $p_i(\boldsymbol{c}^1, \boldsymbol{c}^2)$ be the size of the nondominated set for the first i objects. Let $p(\boldsymbol{c}^1, \boldsymbol{c}^2) \coloneqq \max_{i\in[n]} p_i(\boldsymbol{c}^1, \boldsymbol{c}^2)$, i.e., the size of the largest set we generate. We can also define $p(n) \coloneqq \max\{p(\boldsymbol{c}^1, \boldsymbol{c}^2) \mid \boldsymbol{c}^1, \boldsymbol{c}^2 \in \mathbb{Q}^n\}$, which is the size of the largest set generated over all instances with n objects. Ackermann et al. [1] prove that the expected growth of $p(n)$ is polynomial in the input size in the smoothed-analysis framework (see also Section 4.3). But it is unknown how these $p(n)$ can grow in the worst case with respect to the input and the output.

Moreover, we observe that the order in which the objects are given does not matter in expressing a given instance and has no influence on the nondominated set. But it

makes a difference for the nondominated sets of the first i objects in the Nemhauser-Ullmann algorithm. There might be a permutation $\pi : [n] \to [n]$ of the objects of a given instance which reduces the sizes of the intermediate nondominated sets significantly. The above definition of $p(n)$ takes the worst order into account. Let us thus define $q_i(\boldsymbol{c}^1, \boldsymbol{c}^2, \pi)$ as the size of the nondominated set for the first i objectives when applying the permutation π on the order of items before the Nemhauser-Ullmann algorithm runs. Let $q(\boldsymbol{c}^1, \boldsymbol{c}^2, \pi) \coloneqq \max_{i \in [n]} q_i(\boldsymbol{c}^1, \boldsymbol{c}^2, \pi)$ analogously to $p(\boldsymbol{c}^1, \boldsymbol{c}^2)$. Now we can define the minimum generated set size for all permutations of the objects for an instance as $q(\boldsymbol{c}^1, \boldsymbol{c}^2) = \min_{\pi \in S_n} q(\boldsymbol{c}^1, \boldsymbol{c}^2, \pi)$ and the maximum generated set size for a given instance size when a best permutation is considered: $q(n) = \max\{q(\boldsymbol{c}^1, \boldsymbol{c}^2) \mid \boldsymbol{c}^1, \boldsymbol{c}^2 \in \mathbb{Q}^n\}$.

If we can prove that $q(n)$ grows super-polynomially in the input and the output size then the Nemhauser-Ullmann algorithm is not output-sensitive for BUCO for any permutation of the input objects. Proving that $q(n)$ grows only polynomially in the input and the output size leaves us with the problem of computing an optimal permutation π or one which is good enough, but will come in useful as we see in the next section. If we prove that $p(n)$ grows super-polynomially in the input and the output size, then there still might be a better permutation which might be computable in an output-sensitive way. Otherwise, if we prove that $p(n)$ grows in a polynomial way in the input and output size, we are done and the Nemhauser-Ullmann algorithm is output-sensitive.

The ε-constraint Method

Another solution method is the ε-constrained method. Let us assume w.l.o.g. that $\boldsymbol{c}^1, \boldsymbol{c}^2 \in \mathbb{Z}^n$ to make this argumentation more focused. We solve problems as described in Section 4.4 by using the ε-constraint scalarization. We start by finding an optimal solution $\boldsymbol{x}^*$ for the lexicographic variant of the problem: $\operatorname{lexmin}\{(\boldsymbol{c}^{1T}\boldsymbol{x}, \boldsymbol{c}^{2T}\boldsymbol{x}) \mid \boldsymbol{x} \in \{0,1\}^n\}$. We can then solve the ε-constraint problem

$$\operatorname{lexmin}\{(\boldsymbol{c}^{1T}\boldsymbol{x}, \boldsymbol{c}^{2T}\boldsymbol{x}) \mid \boldsymbol{x} \in \{0,1\}^n, \boldsymbol{c}^{2T}\boldsymbol{x} \leq \boldsymbol{c}^{2T}\boldsymbol{x}^* - 1\}.$$

Then we iterate: We take the solution to this problem as the new $\boldsymbol{x}^*$ and adapt the ε-constraint in each iteration. We can observe, that we only solve one such ε-constraint problem for every nondominated point. But how hard are these ε-constraint problems? We can observe, that these are special cases of the knapsack problem, which is **NP**-hard in general. But since the instances we solve here are a strict subset of the possible knapsack instances, the complexity is not clear.

A possible direction is to prove that the specialized ε-constraint problems are solvable in polynomial time in general or for special cases.

6.2.2. Approaches to Proving Hardness

Departing from the section on the hardness of the MOSP problem, to prove hardness of the BUCO problem, we could investigate the finished decision variant of the BUCO

problem. First we can observe that $BUCO^{\text{FIN}}$ is in **co-NP**: Let's assume we have an instance represented by vectors $\boldsymbol{c}^1, \boldsymbol{c}^2 \in \mathbb{Z}^n$ and a set of vectors $M \in \mathbb{Z}^2$. If someone presents us a solution $\boldsymbol{x} \in \{0,1\}^n$, we can in polynomial time compute its value $\boldsymbol{y} = (\boldsymbol{c}^{1^T}\boldsymbol{x}, \boldsymbol{c}^{2^T}\boldsymbol{x})^T$. Also in polynomial time, we can check whether $\boldsymbol{y} \notin M$ and if $\boldsymbol{y}$ is nondominated in M. If this is the case, $\boldsymbol{y}$ is a certificate that we are not done. In conclusion, we have a proof encoded with n bits for not being finished and thus $BUCO^{\text{FIN}} \in$ **co-NP**. To show that $BUCO^{\text{FIN}}$ is in **NP**, we have to find a proof of polynomial size whether a given set is already the whole nondominated set and we have to be able to verify the proof in polynomial time. One way to do that is to use the Nemhauser-Ullmann algorithm in the proof of the following lemma:

Lemma 6.5. *If* $q(n) \in \text{poly}(n, |\mathcal{Y}_N|)$ *then* $BUCO^{\text{FIN}} \in$ ***NP***.

Proof. Since $q(n) \in \text{poly}(n, |\mathcal{Y}_N|)$, we know that there is a permutation $\pi^* : [n] \to [n]$ such that the Nemhauser-Ullmann algorithm is output-sensitive when applying π^* on the order of the objects in the input. Let the running time be $p(n, |\mathcal{Y}_N|)$. To construct an **NP**-algorithm we have access to a special string y from which we can read at most polynomially many bits. We check if it encodes a permutation $\pi : [n] \to [n]$; if it does not, we output "no". If it does, we permute the objects in the input according to the permutation encoded by y and run $p(n, |M|)$ steps of the Nemhauser-Ullmann algorithm. If it terminates with output M', we check whether $M = M'$ and return "yes" if it does and "no" if it does not. If it does not terminate, we output "no".

Now we have to prove that for an instance x in $BUCO^{\text{FIN}}$, there is a y such that the algorithm returns "yes". Since $q(n) \in \text{poly}(n, |\mathcal{Y}_N|)$, we see that there is one y which encodes π^* and with this permutation the Nemhauser-Ullmann algorithm returns the nondominated set in time $\text{poly}(n, |\mathcal{Y}_N|)$. Since $x \in BUCO^{\text{FIN}}$ it is $M = \mathcal{Y}_N$. Thus the Nemhauser-Ullmann algorithm terminates and we check whether $M = \mathcal{Y}_N$ and return "yes".

On the other hand, if $x \notin BUCO^{\text{FIN}}$ the algorithm has to return "no" for every y. So let us assume that $M \neq \mathcal{Y}_N$. The Nemhauser-Ullmann algorithm does either not terminate or it terminates and returns $\mathcal{Y}_N$. If it does not terminate, we correctly return "no". If it terminates, we correctly return "no" after verifying that $M \neq \mathcal{Y}_N$. The verification can be done in time $\text{poly}(n, |M|)$, since $\mathcal{Y}_N$ can be at most of size $p(n, |M|)$ because that was the maximum running time of the Nemhauser-Ullmann algorithm. □

If $BUCO^{\text{FIN}}$ is in **NP** ∩ **co-NP** it is not very probable that it is also **NP**- or **co-NP**-hard, because this implies **NP** = **co-NP** and thus the collapse of the polynomial hierarchy to the first level, i.e., **NP** = **PH**. Thus we can still try to prove that $BUCO^{\text{FIN}}$ is **NP**-hard, but if $q(n) \in \text{poly}(n, |\mathcal{Y}_N|)$ then this method might most probably not work.

There is also another method we can potentially use to prove that BUCO is not solvable in an output-polynomial way. Let us assume it was solvable in output-polynomial time by an algorithm $\mathcal{A}$ with running time $p(n, |\mathcal{Y}_N|)$. We can use this algorithm to solve the knapsack problem. Let us define for a knapsack instance

$\boldsymbol{c}^1, \boldsymbol{c}^2 \in \mathbb{N}^n$ the nondominated set of the corresponding BUCO instance with input $-\boldsymbol{c}^1$ and $\boldsymbol{c}^2$ as the nondominated set of the knapsack instance. Our algorithm $\mathcal{A}$ now is capable of solving knapsack instances with polynomial sized nondominated sets in polynomial time.

Corollary 6.6. *If there is an **NP**-hard set of knapsack instances with polynomial sized nondominated set, then BUCO $\notin$ **TotalP** unless **P** = **NP**.*

A third way to show hardness of the BUCO problem can go through fixed-parameter tractability. We can investigate the knapsack problem parameterized with the size of its nondominated set. The decision version of the knapsack problem is the following language:

$$\mathrm{KP} = \{\left(\boldsymbol{c}^1, \boldsymbol{c}^2, k_1, k_2\right) \in \mathbb{Q}^n \times \mathbb{Q}^n \times \mathbb{Q}^2 \mid \exists \boldsymbol{x} \in \{0,1\}^n : \boldsymbol{c}^{1^T}\boldsymbol{x} \geq k_1, \boldsymbol{c}^{2^T}\boldsymbol{x} \leq k_2\}$$

Then the parameterized knapsack problem is the following parameterized language:

$$\mathrm{KP}^{\mathrm{P}} = \left\{ (x, M) \in \mathrm{KP} \times \mathbb{N} \mid M = |\min\{ \left(\boldsymbol{c}^{1^T}\boldsymbol{x}, \boldsymbol{c}^{2^T}\boldsymbol{x}\right) \mid \boldsymbol{x} \in \{0,1\}^n\}| \right\}$$

If we have a **TotalP**-algorithm for BUCO we can decide KP^{P} in time $\mathrm{poly}(n, M)$. That means there is a $k \in \mathbb{N}$, such that the running time is bounded by $\mathcal{O}(n^k + M^k)$ which proves that $\mathrm{KP}^{\mathrm{P}} \in \mathbf{FPT}$.

Lemma 6.7. *If BUCO $\in$ **TotalP** $\implies$ $KP^{\mathrm{P}} \in$ **FPT**.*

We can use this in the following way: If we can prove that KP^{P} is $\mathbf{W}[1]$-hard, then *BUCO* $\notin$ **TotalP** unless $\mathbf{FPT} = \mathbf{W}[1]$.

Let us summarize the findings in this section: We can show hardness of the BUCO problem either by finding an **NP**-hard set of knapsack instances with small nondominated set, showing that the parameterized version of the knapsack problem is $\mathbf{W}[1]$-hard or by showing that the finished decision problem is **NP**-hard (or **co-NP**-hard). The latter being not so probable if we can prove that $q(n)$ is bounded by a polynomial in the input and the output size, although this does not prove *BUCO* $\in$ **TotalP**.

6.2.3. BUCO-hard Problems

In the last section we have seen several approaches to the BUCO problem. New results for solving the BUCO problem imply new methods in the field of multiobjective combinatorial optimization. But we can also use the BUCO problem the other way around: If we show that the BUCO problem is a hard enumeration problem, then many interesting MOCO problems turn out to be not solvable in an output-sensitive way.

To investigate this formally we define an output-sensitive reduction which is not invented by us but used more or less formally in many publications.

Definition 6.2 (Output-Sensitive Reduction)**.** For two enumeration problems $P_1 = (\mathcal{I}_1, \mathcal{C}_1)$ and $P_2 = (\mathcal{I}_2, \mathcal{C}_2)$, an *output-sensitive reduction* from P_1 to P_2, or

$$P_1 \leq_{\mathbf{TotalP}} P_2,$$

is a polynomial-time computable function

$$f \colon \mathcal{I}_1 \to \mathcal{I}_2,$$

with

$$\mathcal{C}_2(f(x)) = \mathcal{C}_1(x).$$

The purpose of such a reduction is to transfer hardness results. But this only works if hardness of one problem implies hardness of the other problem, or equivalently, the output-sensitive solvability of one problem implies the output-sensitive solvability of the other. We prove this now for the output-sensitive reduction we defined above.

Proposition 6.8. *Let $P_1 = (\mathcal{I}_1, \mathcal{C}_1)$ and $P_2 = (\mathcal{I}_2, \mathcal{C}_2)$ be enumeration problems and $P_1 \leq_{\boldsymbol{TotalP}} P_2$. Then $P_2 \in$ **TotalP** $\implies$ $P_1 \in$ **TotalP**.*

Proof. Let $\mathcal{A}$ be a **TotalP**-algorithm for P_2, we now construct a **TotalP**-algorithm for P_1. We get an instance $x_1 \in I_1$ of P_1 and use the polynomial-time algorithm—which exists by definition of $\leq_{\mathbf{TotalP}}$—to compute $x_2 \coloneqq f(x_1) \in I_2$ in time $p(|x_1|)$ for a polynomial p. Then we let $\mathcal{A}$ run on x_2 with output $\mathcal{C}_2(x_2)$ in time $q(|x_2|, |\mathcal{C}_2(x_2)|)$ for a polynomial q. Consequently, we output $\mathcal{C}_2(x_2)$.

Since $P_1 \leq_{\mathbf{TotalP}} P_2$, it holds that $\mathcal{C}_2(x_2) = \mathcal{C}_2(f(x_1)) = \mathcal{C}_1(x_1)$ and thus the output is correct. Now we investigate the running time: First we construct the instance x_2 which takes time $p(|x_1|)$, thus also $|x_2| \leq p(|x_1|)$. Then we let $\mathcal{A}$ run with running time $q(|x_2|, |\mathcal{C}_2(x_2)|) \leq q(p(|x_1|), |\mathcal{C}_1(x)|)$ which is output-polynomial for problem P_1. □

We note that the definition of $\leq_{\mathbf{TotalP}}$ is very strong in the sense that we require that $\mathcal{C}_2(f(x)) = \mathcal{C}_1(x)$. We could arrive at a similar result as Proposition 6.8 by allowing $\mathcal{C}_2(f(x))$ to be modified in order to arrive at $\mathcal{C}_1(x)$. But for the following reductions in this section the stronger version suffices.

We now consider two problems of popular interest for which we were not able to pinpoint the output-sensitive complexity. We show that they are hard enumeration problems if the BUCO problem is a hard enumeration problem by using output-sensitive reductions. The first problem being the multiobjective minimum spanning tree problem (MOST, cf. Definition 2.16) and the second one being the multiobjective assignment problem (MOAP, cf. Definition 2.18).

Proposition 6.9. *BUCO $\leq_{\boldsymbol{TotalP}}$ MOST*

Proof. We first have to find a function f from the instances of the BUCO problem to the instances of the MOST problem computable in polynomial time. Let thus an instance of the BUCO problem be given as $\boldsymbol{c}^1, \boldsymbol{c}^2 \in \mathbb{Q}^n$. We construct an undirected graph $G = (V, E)$ and an edge-weight function $c \colon E \to \mathbb{Q}^2$ as an instance of the MOST

problem in the following way: We create $n+1$ nodes and call them v_i for $i \in [n+1]$ and connect them with edges $\{v_i, v_{i+1}\}$ for every $i \in [n]$ and assign cost of $(\boldsymbol{c}_i^1, \boldsymbol{c}_i^2)^T$. We also create nodes w_i for $i \in [n]$ and connect them with two edges $\{v_i, w_i\}$ and $\{w_i, v_{i+1}\}$ and assign cost $\mathbf{0}$ to both edges.

Now we observe that every so constructed instance is a valid MOST instance and that the construction takes time in $\mathcal{O}(n)$. In order to show that the reduction is valid, we now have to prove that the nondominated set of the constructed MOST instance is exactly the nondominated set of the BUCO instance we started with. But we prove something stronger: We prove that the value vectors of the BUCO instance are exactly the value vectors of the constructed MOST instance.

So let $\boldsymbol{y}$ be a value vector of a BUCO instance given by $\boldsymbol{c}^1, \boldsymbol{c}^2 \in \mathbb{Q}^n$. Let $\boldsymbol{x}$ be a solution that is mapped to $\boldsymbol{y}$, i.e., $\boldsymbol{y} = (\boldsymbol{c}^{1T}\boldsymbol{x}, \boldsymbol{c}^{2T}\boldsymbol{x})^T$. There is a path in G from node v_1 to node v_{n+1} with the following property: For every $i \in [n]$, if $x_i = 1$, we take edge $\{v_i, v_{i+1}\}$ and if $x = 0$, we take edges $\{v_i, w_i\}$ and $\{w_i, v_{i+1}\}$. For every $x_i = 1$ the node w_i does not occur in the path and hence the path is not a spanning tree. But by adding for each such w_i the edge $\{w_i, v_{i+1}\}$ with cost $\mathbf{0}$, we have constructed a spanning tree T of G. The cost of the spanning tree is

$$c(T) = \sum_{e \in T} c(e) = \sum_{i \in [n]: x_i = 1} \begin{pmatrix} \boldsymbol{c}_i^1 \\ \boldsymbol{c}_i^2 \end{pmatrix} = \boldsymbol{y}.$$

Now, let $\boldsymbol{y}$ be a value vector of a constructed MOST instance from a BUCO instance represented by $\boldsymbol{c}^1, \boldsymbol{c}^2 \in \mathbb{Q}^n$. Let $T \subseteq E$ a spanning tree with $\boldsymbol{y} = c(T) = \sum_{e \in T} c(e)$. We now construct a solution $\boldsymbol{x} \in \{0,1\}^n$ for the BUCO instance: We set $\boldsymbol{x}_i = 1$, if T contains the edge $\{v_i, v_{i+1}\}$ and $\boldsymbol{x}_i = 0$ otherwise. The cost of such a solution is

$$\begin{pmatrix} \boldsymbol{c}_1{}^T\boldsymbol{x} \\ \boldsymbol{c}_2{}^T\boldsymbol{x} \end{pmatrix} = \sum_{i \in [n]: \boldsymbol{x}_i = 1} c(\{v_i, v_{i+1}\}) = \sum_{e \in T} c(e) = c(T) = \boldsymbol{y},$$

since all edges apart from the $\{v_i, v_{i+1}\}$-edges have cost $\mathbf{0}$. □

We can prove a similar result for the multiobjective assignment problem:

Proposition 6.10. *$BUCO \leq_{\boldsymbol{TotalP}} MOAP$*

Proof. We again have to find a function which maps an instance of the BUCO problem to an instance of the MOAP problem. Let an instance of the BUCO problem be given with $\boldsymbol{c}^1, \boldsymbol{c}^2 \in \mathbb{Q}^n$. We construct a complete bipartite graph $G = (V, E)$ and an edge-cost function $c\colon E \to \mathbb{Q}^2$. We add $4n$ nodes to G: For each $i \in [n]$ we add node v_i, nodes w_i^1 and w_i^2 and node u_i. For each $i \in [n]$, we also add the edges $\{v_i, w_i^1\}$ and $\{v_i, w_i^2\}$. The first edge gets cost $(\boldsymbol{c}_i^1, \boldsymbol{c}_i^2)$ and the other gets cost $\mathbf{0}$. Wo connect all the u_i nodes to all nodes w_i^1 and w_i^2 with cost $\mathbf{0}$.

We again prove the stronger result that every value vector of the BUCO instance is also a value vector of the so constructed MOAP instance and vice versa. Let thus $\boldsymbol{y}$ be a value vector of a BUCO instance given by $\boldsymbol{c}^1, \boldsymbol{c}^2 \in \mathbb{Q}^n$ and let $\boldsymbol{x}$ be a solution

that is mapped to $\boldsymbol{y}$, i.e., $\boldsymbol{y} = ({\boldsymbol{c}^1}^T\boldsymbol{x}, {\boldsymbol{c}^2}^T\boldsymbol{x})^T$. We can construct a solution matching M to the corresponding MOAP instance by choosing edge $\{v_i, w_i^1\}$ if $\boldsymbol{x}_i = 1$ and edge $\{v_i, w_i^2\}$ if $\boldsymbol{x}_i = 0$. We observe that no node v_i is free anymore, but some nodes w_i^1 and w_i^2 and all of the u_i nodes are free. We can now add the remaining edges incident to the u_i nodes to make all the remaining unmatched nodes w_i^1 and w_i^2 covered and the matching perfect. The cost of the matching is

$$\sum_{e \in M} c(e) = \sum_{i \in [n]: x_i = 1} \begin{pmatrix} \boldsymbol{c}_i^1 \\ \boldsymbol{c}_i^2 \end{pmatrix} = \boldsymbol{y}.$$

Let now M be a perfect matching in the constructed graph G from a BUCO instance represented by $\boldsymbol{c}^1, \boldsymbol{c}^2 \in \mathbb{Q}^n$. We construct a solution $\boldsymbol{x}$ to the BUCO instance in the following way: If M contains the edge $\{v_i, w_i^1\}$ for an $i \in [n]$, we set $\boldsymbol{x}_i = 1$, or if not, we set $\boldsymbol{x}_i = 0$. The cost of this solution $\boldsymbol{x}$ thus is

$$\sum_{i \in [n]: \boldsymbol{x}_i = 1} \begin{pmatrix} \boldsymbol{c}_i^1 \\ \boldsymbol{c}_i^2 \end{pmatrix} = \sum_{e \in M} c(e) = c(M),$$

since all edges but those for which we set $\boldsymbol{x}_1 = 1$ have cost $\mathbf{0}$. □

The observation of problems that turn out to be not solvable in output-polynomial time if the BUCO problem cannot be solved in output-polynomial time gives rise to a special hardness notion. We use a similar notion as also used for graph isomorphism, i.e., **GI**-hardness, and vertex enumeration, i.e., **VE**-hardness.

Definition 6.3. The set of MOCO problems O for which $O \leq_{\mathbf{TotalP}} BUCO$ holds is called **BUCO**. We call a problem **BUCO**-hard, if it is **BUCO**-hard with respect to output-sensitive reduction.

And we can formulate the result of the previous propositions in a succinct form:

Corollary 6.11. *MOST and MOAP are **BUCO**-hard.*

7. Conclusion

Let us review Table 3.1 in the light of the past two chapters. In the "Classic Complexity" column, we have seen in Section 4.1 that there are doubts if problems as the MOST, MOAP and the BUCO problem are **NP**-hard. Whereas in Section 6.1, we have seen that the MOSP problem is indeed **NP**-hard. Through this reasoning, we also get the **NP**-hardness of MO-Z-Flow and MO-Matching.

In the column "Nondominated Set", we were able to fill almost all the gaps there. For the MO-TSP problem, the proof is the same as we prove that the problem is **NP**-hard and the result transfers to the MOCO problem since MO-TSP is a special case. The result for MOLP^{XEx} comes from the work by Khachiyan et al. [85]. We can reduce the problem of finding extreme points of a polyhedron to the problem of finding all Pareto-optimal extreme points of an MOLP with only two objectives. The results on MOLP^{YEx} and MOLP^{YFa} are the central results of Chapter 5. But these results come with two assumptions: First, an ideal point must exist and second, the number of objectives must be a fixed number.

That the MOSP is a hard enumeration problem is one of the main results of Section 6.1. The consequences for practitioners were already stated in the introduction of this thesis. Again, this transfers to the MO-Z-Flow and MO-Matching problem, because MOSP is a special case. For the BUCO problem, we gave reason that an output-sensitive algorithm for BUCO would need new algorithmic ideas for multiobjective combinatorial optimization in general and if the BUCO problem is a hard enumeration problem, several other MOCO problems are hard enumeration problems. This gave rise to the complexity class **BUCO** and we coined two major **BUCO**-hard problems: The MOST problem and the MOAP problem. The result for the MOMC problem can be directly derived from the literature and this is explained in the preliminaries chapter in Section 4.4.

Concerning the last column "Extreme Points", most of these results come from the extensive theory in Chapter 5 which culminate in the results for MOCO problems in Section 5.4. Since all of the problems MO-Matching, MO-Z-Flow (the totally unimodular version from the definition), MOAP, MOSP, MOST, BUCO, and MOMC have polynomially solvable weighted-sum problems, we can enumerate their extreme points in output-polynomial time. Because the MO-Z-Flow problem and the MOAP problem have compact LP formulations, the extreme points can even be enumerated in incremental polynomial time. Concerning the MOSP, MOST, BUCO, and MOMC problem, we have seen that the number of extreme points grows polynomially in the input size for every fixed number of objectives so that the extreme points can even by enumerated in polynomial time for every fixed number of objectives.

Open Problems

Concerning output-sensitive complexity and MOCO problems in general, the BUCO problem is the open problem which probably will generate many new insights. It is also a barrier, since we cannot expect to make positive progress on the MOAP or MOST problem without also progressing considerably on the BUCO problem. Also finding more problems in **BUCO** and **BUCO**-complete problems can lead to understanding the BUCO problem better.

Concerning MOLP, it is interesting if we can get rid of the assumption that an ideal point exists. The challenge is that we need to find an outer approximate polyhedron without knowing the upper image itself. Even finding a first extreme point can be challenging, since the weights with finite weighted-sum LPs are not known beforehand. Fixing the number of objectives is something we most probably cannot get rid of, because of the relation to the projection of cones and polyhedra (cf. Löhne and Weißing [96]) which in turn is an **NP**-hard enumeration problem (cf. Tiwary [133]). It is also interesting if the running time itself can be improved. Is polynomial delay possible for each fixed number of objectives? This cannot be ruled out by what we know at the moment and constitutes a major challenge.

Linear programming in single objective optimization is an essential part in many approximation algorithms for single-objective combinatorial optimization problems. A hope is that we can find good approximation algorithms for MOCO problems through being able to solve MOLP relaxations. This is especially interesting for practice since most of the approximation algorithms designed for MOCO problems depend exponentially on the number of objectives even in the best case.

Another interesting and rather young field is fixed parameter enumeration (cf. Creignou et al. [37]). Analogously to fixed parameter tractability, the question is if we can isolate certain parameters of instances of a MOCO problem to allow an exponential running time in this parameter but polynomial in the size of the nondominated set and the input. Can we find a parameter interesting for MOLP and thus find an algorithm with parameterized polynomial delay? Is there an interesting parameter such that we can find a parameterized output-polynomial time algorithm for the MOSP problem?

Part II.

The Multiobjective Shortest Path Problem in Practice

8. Introduction

In this second part, we turn to a more practical problem. As typical for Algorithm Engineering, the most successful result can be reaped if the implementation is tailored to the practical problem. Thus in this part, we want to approach the problem of finding "good" power transmission lines.

In Germany, after the events of the Fukushima catastrophe in March of 2011, it was decided that nuclear power plants should be shut down by 2022[1]. At the time of this writing, of the seven existing nuclear power plants[2], four reside in the south of Germany—Philippsburg and Neckarwestheim in Baden-Württemberg, and Gundremmingen and Isar in Bavaria. The fact that regenerative energy producers are not as centralized as nuclear power plants are, gives rise for the need for power transmission lines; often even from far in the north to far in the south (cf., e.g., SuedLink[3] from Brunsbüttel, Schleswig-Holstein, to Großgartach, Baden-Württemberg). The "Netzentwicklungsplan 2022" or power grid development plan 2022 describes 2,800km of lines to be built and 2,900km of power transmission lines to be reinforced[4].

In a research project by the German Federal Ministry for Economic Affairs and Energy together with the spatial planning chairs "Ver- und Entsorgunssysteme" and "Raumbezogene Informationsverarbeitung und Modellbildung", as well as the chair for Discrete Optimization in mathematics and the computer science chairs computer graphics and algorithm engineering and the spatial planning consulting bureau "Spiekermann & Wegener Stadt- und Regionalforschung" as well as the Amprion GmbH, we were investigating many aspects of this problem. We decided to model the problem of finding new power transmission line alternatives between two points as a multiobjective shortest path problem. Especially the number of criteria to be considered turned out to be a major computational burden.

From the literature and first part of this thesis we learn the following statements about the MOSP problem:

[1] https://www.bmwi.de/Redaktion/DE/Downloads/E/energiekonzept-2010-beschluesse-juni-2011.pdf, BMWi (German Federal Ministry for Economic Affairs and Energy), last accessed: Dec. 22nd 2017.

[2] http://www.bmub.bund.de/themen/atomenergie-strahlenschutz/nukleare-sicherheit/aufsicht-ueber-kernkraftwerke/kernkraftwerke-in-deutschland/, BMUB (German Federal Ministry for the Environment, Nature Conservation, Building and Nuclear Safety), last accessed: Dec. 22nd 2017.

[3] https://www.tennet.eu/de/unser-netz/onshore-projekte-deutschland/suedlink/erdkabel-korridore/korridorverlaeufe/topographische-regionalkarten/, Tennet (German Transmission Grid Maintainer), last accessed: Dec. 22nd 2017.

[4] Netzentwicklungsplan 2022: 50Hertz Transmission, Amprion, TenneT TSO, TransnetBW (https://www.netzausbau.de/bedarfsermittlung/2022/nep-ub/de.html), last accessed: Dec. 22nd 2017.

- The nondominated set can be of exponential size. (Hansen [74])
- The problem is **NP** hard. (Proposition 6.4)
- We expect the output to not be too large when the number of objectives is small. (Brunsch and Röglin [26])
- There is no output-sensitive algorithm for the MOSP problem, unless $\mathbf{P} = \mathbf{NP}$. (Theorem 6.2)
- The nondominated set can be computed in smoothed polynomial time if the number of objectives is not too large. (Using Ackermann et al. [1] and, e.g., Martins [98])

As a consequence, while modeling the problem, the number of criteria needed to be reduced as much as possible. Starting with more than 200 criteria that was not easily possible and a reduction to less than eight or nine criteria would result in a loss of too much information.

It became apparent early in the project, that with that many criteria an exact approach would be prohibitive. So a good approximative method was sought. It turned out, as we see in Chapter 10, that the running times of the approximation algorithms available have worst-case running-times bounded exponentially in the number of objectives. And also the best-case running-time on non-trivial instances happens to be exponentially bounded from below in the number of objectives. Consequently, a major challenge in this project from an algorithm engineering point of view was to find an approximation algorithm with a running-time less sensitive to the number of objectives.

8.1. Contributions and Organisation

The second part is divided into two chapters: In Chapter 9, we consider preliminaries regarding the MOSP problem. Our special interest are labeling algorithms and we discuss the literature about these algorithms. We also introduce the instance set for all the experiments in this part which comes from several sources. In the same chapter we consider a heuristic speeding up label correcting algorithms and investigate its performance in practice.

In Chapter 10, we then introduce multiobjective approximation and give an overview on the literature. The core of this chapter is the description of a new approximation algorithm which is much less dependent on the number of objectives and uses the labeling techniques and heuristic from Chapter 9. We also present extensive computational experiments to put the new algorithm into the context of the literature.

9. Preliminaries

Since the first paper by Hansen [74], the MOSP problem gained much interest from the scientific community. The majority of the literature is concerned with labeling algorithms and we discuss the general methodology in Section 9.1. Also, the majority of the literature deals with the case of 2 objectives and tailoring special labeling algorithms to this case. We are explicitly not dealing with the biobjective case and only consider instances with at least 3 objectives, which is motivated from our application.

There exist recent computational studies dealing with the efficiency of labeling algorithms. But there is one problematic issue with these studies: The source code is not available. This renders a comparison of the exact implementations of recent studies impossible and the algorithm variants need to be implemented from scratch. Since implementations cannot be compared, it is also hard to say which general method is more promising, because a difference in the performance of an implementation can be solely due to implementation reasons.

We make our implementations available online[1]. We hope that this makes it easier for comparisons and to lay ground for the study of when which algorithmic variant is preferable.

The instance sets we use in our study are also available from various sources. An overview on the instances we use and where to get them is given in Section 9.2.

In Section 9.3, we compare two major strategies for label correcting algorithms: the node-selection and label-selection strategy. We show that contrary to the results in the literature, node-selection is always faster than label-selection strategies.

Then in Section 9.4, we introduce a heuristic for label-correcting algorithms, the tree-deletion heuristic. We show that it is useful in many cases, especially in real-world scenarios. The heuristic becomes an integral part of our approximation algorithm.

9.1. Multiobjective Labeling Algorithms

There are many variants of multiobjective labeling algorithms for the MOSP problem, but all of these methods share a similar idea: The algorithms maintain a set of labels L_u at each node $u \in V$. A label is a tuple which represents a path from s to a node $u \in V$ and it consists of the cost vector of the path, the associated node u and, for retrieving the actual path, a reference to the predecessor label. The algorithms are initialized by setting each label-set to $\emptyset$ and adding the label $(\mathbf{0}, s, \mathbf{nil})$ to L_s.

[1]The code is available on GitHub: https://github.com/FritzBo/mco

Labeling algorithms for the MOSP problem can be divided into two classes, depending if they select either a label or a node in each iteration. These strategies are called *label* or *node selection strategies*, respectively. When we select a label $\ell = (\mathbf{v}, u, \hat{\ell})$ at a node u, this label is *pushed* along all out-arcs $a = (u, w)$ of u, meaning that a new label is created at the head of each arc with cost $\mathbf{v} + c(a)$, predecessor label ℓ and associated node w. This strategy is due to Tung and Chew [135] for $d > 2$. If we follow a node-selection strategy, all labels in L_u are pushed along the out-arcs of a selected node u. This strategy was first proposed in Brumbaugh-Smith and Shier [25] for the biobjective case and has been generalized by many authors in subsequent works. Nodes or labels that are ready to be selected are called *open*.

After pushing a set of labels, the label sets at the head of each considered arc are *cleaned*, i.e. all *dominated* labels are removed from the modified label sets. We say a label $\ell = (\mathbf{v}, u, \hat{\ell})$ dominates a label $\ell' = (\mathbf{v}', u', \hat{\ell}',)$ if $\mathbf{v} \preceq \mathbf{v}'$. Moreover, if a label already exists at the target node, usually the new label is discarded and the old one is kept.

There are many ways in which an open label or node can be selected. A comparative study was conducted by Paixao and Santos [107]. For example, a pure FIFO strategy seems to work best in the aforementioned study. But also other strategies are possible: For example, we can sort the labels by their average cost, i.e., $\sum_{i \in \{1,\dots,d\}} \mathbf{v}_i / d$ and always select the smallest one. A less expensive variant is due to Bertsekas, Guerriero, and Musmanno [15] and Paixao and Santos [108], where we decide depending on the top label $\ell = (\mathbf{v}, u, \hat{\ell})$ in a FIFO queue Q where to place a new label $\ell' = (\mathbf{v}', u', \hat{\ell}')$: If $\mathbf{v}'$ is lexicographically smaller than $\mathbf{v}$, then it is placed at the front of Q, otherwise it is placed at the back of Q.

Another result from the available computational studies on labeling algorithms on problems with more than 2 criteria suggests that label-selection strategies are far superior compared to node-selection strategies [71, 108].

After all labels are pushed and there are no open labels left, we can reconstruct the paths connected to the labels if we want to. This can be achieved by recursively following the previous label pointer until we reach the initial label at the source node.

Label-Setting vs. Label-Correcting Algorithms

There is a special method based on a label-selection strategy which is due to Martins [98]. The algorithm selects the next label by choosing the lexicographically smallest label among all labels in $\mathcal{L} \coloneqq \bigcup_{u \in V} L_u$. In general, whenever we select a lexicographically smallest label $\ell = (\boldsymbol{v}, u, \hat{\ell})$ in $\mathcal{L}$, this label represents a nondominated path in $\mathcal{P}_{s,u}$. Labeling algorithms having the property that whenever we select a label we know that the represented path is a Pareto-optimal path, are called *label-setting algorithms*. Labeling algorithms which do not have this property, and thus sometimes delete or correct a label, are called *label-correcting algorithms*. On the plus side, in label-setting algorithms, we never select a label which is deleted in the process of the algorithm. But selecting these labels is not trivial. For example, selecting a lexicographically smallest label requires a priority queue data structure, whereas the simplest label-selection

strategy requires only a simple FIFO queue.

There is recent work by Erb, Kobitzsch, and Sanders [61], which suggests that in the biobjective case, a label-setting approach can outperform the label-correcting method they considered. But in the studies for 3 and more objectives, the label-correcting algorithms are superior to the label-setting variants [71, 108], even when considering several heap data structures [108]. It is not clear why this is the case. One possible reason for this is that the cost of the data structure can not make up for the advantage of not pushing too many unneeded labels.

Computational Experiments in the Literature

We now give an overview on the existing studies on labeling methods for the MOSP problem, but we focus only on the studies considering three and more objectives.

The latest computational study for more than 2 objectives is by Paixao and Santos [108] from 2009 and compares 27 variants of labeling algorithms on 9,050 artificial instances. These are the instances we also use for our studies. In summary, a label-correcting implementation with a label-selection strategy in a FIFO manner is concluded to be the fastest strategy on the instance classes provided. The authors do not consider a node-selection strategy with the argument that it is harder to implement and is less efficient (cf. also Paixao and Santos [107]). They also conclude that the label-setting approach is in general less efficient than the label-correcting methods.

In an older study by Guerriero and Musmanno [71] from 2001, also label-selection and node-selection strategies are compared. The authors conclude that, in general, label-selection methods are faster than node-selection methods. However, the test set is rather small, consisting of only 8 artificial grid-graph instances ranging from 100 to 500 nodes and 2 to 4 objectives and 18 artificial random-graph instances ranging from 500 to 40,000 nodes and densities of 1.5 to 30 with 2 to 4 objectives. The authors also conclude that the label-setting approach is slower on larger graph instances.

As the source code of Guerriero and Musmanno [71] and Paixao and Santos [108] is not available, we cannot say how our implementations behave compared to the implementations tested in these studies. But for the sake of simplicity we assume that the general observation is true that the label-correcting methods are superior to the label-setting method if the graphs are sufficiently large or the instances have a sufficiently large number of objectives.

In the work by Delling and Wagner [44], the authors solve a variant of the multiobjective shortest path problem where a preprocessing is allowed and we want to query the nondominated set of paths between a pair of nodes as fast as possible. The authors use a variant of SHARC (SHort cuts and ARC flags, cf. Bauer and Delling [10]) to solve this problem. Though being a different problem, this study is the first computational study where an implementation is tested on real-world road networks instead of artificial instances. The instances have sizes of 30,661, 71,619 and 892,392 nodes and 2 to 4 objectives. Though, the largest instance could only be solved using highly correlated objective functions resulting in nondominated set sizes of only at most 2.5 points on average.

Name	n	d	density
`CompleteN-large`	10–200	6	$n-1$
`CompleteK-large`	100	3–9	$n-1$
`GridN-large`	100–400	6	~4
`GridK-large`	100	3–9	~4
`RandomN-large`	1000–20000	6	6
`RandomK-large`	5000	3–9	6

Table 9.1.: Overview on the artificial test instances by Paixao and Santos [108]

9.2. Instances

We used three sets of instances. First, we took the instances by Paixao and Santos [108] and tested our implementations on them. The instance set consists of random, complete and grid graphs. An overview of these instances can be seen in Table 9.1.

The random graph instances are based on a Hamiltonian cycle and further arcs are randomly added. In the complete graph instances, arcs are added between each pair of nodes in both directions. In the grid graphs, each node is connected to its right and lower neighbor if it exists. The grid graphs are square and the start node is in one corner of the graph and the target node is in the opposite corner of the graph. For all these instances, the arc costs are chosen uniformly at random in $[1, 1000] \cap \mathbb{N}$. For each problem type, there are 50 randomly drawn instances. In summary they make up a set of 9,050 test problems.

The instances come in three size groups: small, medium and large. It turns out that the small and medium size instances are so small that they can be solved in milliseconds and do not show much variation among the implementations. In our tests, we only use the large size instances. The instances are available online, cf. the work by Paixao and Santos [108] for details.

The second set of instances is similar to the instances from a work by Delling and Wagner [44]. They are based on the road network of Western Europe provided by PTV AG for scientific use. We conduct our experiments on the road network of the Czech Republic (CZE, 23,094 nodes, 53,412 edges), Luxembourg (LUX, 30,661 nodes, 71,619 edges), Ireland (IRL, 32,868 nodes, 71,655 edges) and Portugal (PRT, 159,945 nodes, 372,129 edges, only for the tree-deletion experiments). As metrics we use similar metrics as those by Delling and Wagner: travel distance, cost based on fuel consumption and travel time. The median nondominated set sizes are 13.0, 30.5, 12.5 and 133.5, respectively and thus comparable or larger than those by Delling and Wagner. For each of the instances we draw 50 pairs of source and target nodes uniformly at random.

We also test our implementations on a set of instances coming from the finding of Pareto-optimal power transmission lines. The test area is a small 6 km times 3 km square near Münster, North Rhine-Westfalia in Germany. The considered criteria for this first study are bird preservation areas (BPA), landscape preservation areas (LPA),

existing power lines (EPL), freeways (FW), settlement areas (SA), and length (L). We superimpose an undirected grid graph (i.e., there exists an arc in both directions) with eight neighbors per inner node and raster pitch of 100 m on the area we investigate. The resulting graph has 1859 nodes and 12674 edges. For each criterion, we add one objective. In the case of the BCA, LPA, and SA criteria, an edge gets the cost of the length of traversing this area. The settlement area gets a buffer of 400 m which follows German and North Rhine-Westfalian law. For the EPL and FW criteria, the case is the other way around: Instead of trying to avoid these linear infrastructures, it is preferable to build the new line in a 200 m buffer around these. To represent this, we add an area to avoid everywhere outside the 200 m buffer around the infrastructure. Thus, an edge gets the cost of the length of not traversing the buffer. To get different instances, we use different combinations of criteria.

9.3. Node Selection vs. Label Selection

Early in our comparative studies to find the best labeling strategies for our project, we tried several strategies for selecting next labels in label-correcting methods. Our findings and conjectures were that node-selecting strategies are far better than label-selection strategies, while in the literature usually label-selection strategies are described to be superior. It is hard to prove who is right or wrong here since no source-code is available for the computational studies. To make our points attackable by other researchers, we provide the implementations online, the instances are also available online and we describe the implementations in detail.

So to evaluate whether node or label selection is a preferred method, we want to test these variants on our instances. We implemented both label-correcting algorithms, a version of the FIFO label-selection (LS) and FIFO node-selection (NS) algorithms in C++11. The reason for choosing these variants is that the LS-algorithm is the fastest method in the latest comparative study [108].

9.3.1. Implementation Details

A pseudocode overview of the implementations can be found in Listing 3 and 4. We use the OGDF[2] for the representation of graphs, because it is a well tested graph library. For cache efficiency, we try to use `std::vector` for collections of data wherever possible.

Node Selection In the node-selection variants we use our own implementation of a ring buffer based on `std::vector` to implement the queue of open nodes. Also, only those labels of a node are pushed, which have not been pushed before.

Label Selection In the label-selection variant we use a `std::deque` to implement the queue of open labels. A ring buffer cannot be used easily, because the size of the

[2] http://ogdf.net/

Listing 3 Abstract version of the LS algorithm

Require: Graph $G = (V, A)$, nodes $s, t \in V$ and objective function $c\colon A \to \mathbb{Q}^d$
Ensure: List R of pairs (p, y) for all $y \in c(\mathcal{P}_{s,t})$ and some $p \in \mathcal{P}_{s,t}$ such that $c(p) = y$

```
1: L_u ← ∅ for all u ∈ V\{s}
2: ℓ ← (0, s, nil)
3: L_s ← {ℓ}
4: Q.push(ℓ)
5: while not Q.empty do
6:     ℓ = (v, u, ℓ̂) ← Q.pop
7:     for each (u, w) ∈ A do
8:         Push ℓ along (u, w) and add the new label ℓ′ to L_w
9:         Clean L_w
10:        if ℓ′ is nondominated in L_w then
11:            Q.push(ℓ′)
12: Reconstruct paths for each label in L_t and output path/vector pairs
```

Listing 4 Abstract version of the NS algorithm

Require: Graph $G = (V, A)$, nodes $s, t \in V$ and objective function $c\colon A \to \mathbb{Q}^d$
Ensure: List R of pairs (p, y) for all $y \in c(\mathcal{P}_{s,t})$ and some $p \in \mathcal{P}_{s,t}$ such that $c(p) = y$

```
1: L_u = ∅ for all u ∈ V\{s}
2: L_s = {(0, s, nil)}
3: Q.push(s)
4: while not Q.empty do
5:     u ← Q.pop
6:     for each (u, w) ∈ A do
7:         for each not yet pushed label ℓ in L_u do
8:             Push ℓ along (u, w) and add the new label to L_w
9:         Clean L_w
10:        if at least one new label survived the cleaning process and w ∉ Q then
11:            Q.push(w)
12: Reconstruct paths for each label in L_t and output path/vector pairs
```

queue can vary a lot, i.e., it can be as small as 1 or larger than 2^n.

Cleaning Step While there is a considerable literature on finding the subset of minimal vectors in a set of vectors, it is not clear which method to use in practice. There exist practical implementations mostly for the case of 2 or 3 objectives. Implementations for more objectives are usually multiprocessor or GPU based and begin to be efficient only if the data set is very large, i.e., more than 10^5 points (cf., e.g., Bogh, Assent, and Magnani [16] and Mullesgaard et al. [101]), whereas we usually deal with a lot less points (cf., e.g., Table 9.2). The more theoretical results by Kirkpatrick and Seidel [86] and Kung, Luccio, and Preparata [90]—even though quite some time on the market now—do not seem to have found practical use yet. In our preliminary tests it turned out to be the case that when the number of objectives is growing and the number of vectors is not too large, a simple pairwise comparison between the labels works very well especially when tuned to be cache efficient.

9.3.2. Question, Metrics and Experimental Setting

Our main question in this study is whether node or label selection is preferable. Testing both implementations gives us a good idea which one is better, depending on the actual difference in performance.

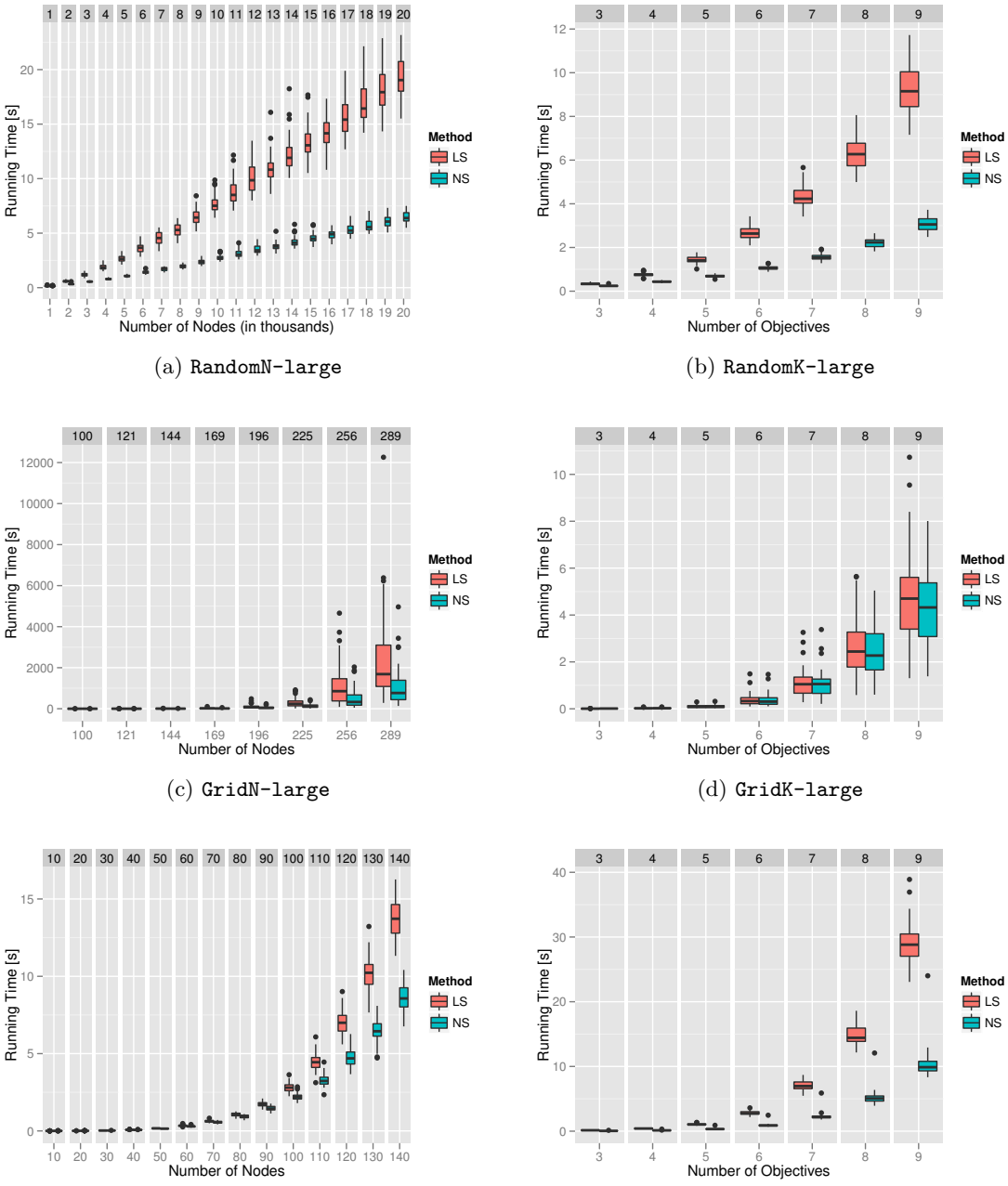
We use the median running times to compare them, since the distribution of running times is expected to be skewed. To gain an overview, we visualize the running times with box plots. The box plots give a direct overview on the quartiles (box dimension) and median (horizontal line inside the box). The size of the box in the y-direction, or the difference of the first and third quartile is called *interquartile range*. The *whiskers* (lines above and below the box) show the range of points which lie below the third quartile plus 1.5 times the interquartile range and above the first quartile minus 1.5 times the interquartile range. Outliers are given by points above and below the whiskers.

The experiments are performed on an Intel Core i7-3770, 3.4 GHz and 16 GB of memory running Ubuntu Linux 12.04. The algorithms are implemented in C++11 and the code is compiled using LLVM 3.4 with compiler flag -O3.

9.3.3. Results

In Figure 9.1 we see the results on the large size instances by Paixao and Santos [108]. We see that the node-selection strategy performs better than the label-selection strategy on all these instances. The measured maximum factor of how much faster the node-selection strategy is compared to the label selection strategy is 3.17 over all instances.

Also on the real-world road networks the results are positive. The results can be seen in Figure 9.2. The node-selection strategy is up to factor of 3.16 faster than the label-selection strategy.



(a) `RandomN-large` (b) `RandomK-large`

(c) `GridN-large` (d) `GridK-large`

(e) `CompleteN-large` (f) `CompleteK-large`

Figure 9.1.: Comparison of the running times (in seconds) of the label-selection (LS) and node-selection (NS) strategies on the instances by Paixao and Santos [108].

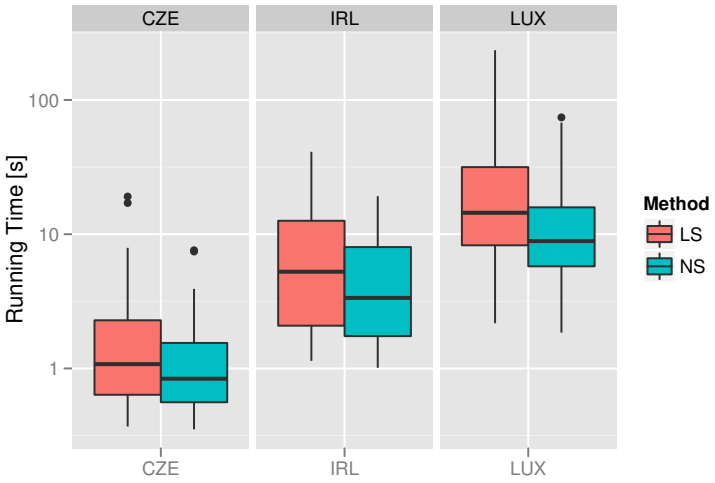


Figure 9.2.: Comparison of the running times (in seconds) of the label-selection (LS) and node-selection (NS) strategies on real-world road networks by PTV AG.

Figure 9.3.: Caption this!

9.3.4. Discussion

After evaluating the figures, it is hard to hold to the thesis that the label-selection strategy is always preferable to the node-selection strategy. This is especially true since the performance gain of the node-selection runs increase as the instance sizes grow for every single instance group.

There is also a good reason why the node-selection strategy performs better when implemented carefully: In the node-selection strategy a consecutive chunk of memory which contains the values of the labels pushed along an arc can be accessed in a small amount of cache accesses. While in the label-selection strategy only one label is picked in each iteration, producing potentially many cache misses when the next label—potentially at a very different location—is accessed.

9.4. Tree-Deletion Pruning

As we discussed the differences between label-correcting and label-setting algorithms previously, we pointed out a major shortcoming of label-correcting algorithms: It can happen that we push labels which later are dominated by a new label. To address this issue, let us take a label-selection algorithm into consideration which selects the next label in a FIFO manner. In Fig. 9.3, we see the situation where a label ℓ at node v, which encodes a path a from s to v, is dominated by a label ℓ', which encodes a

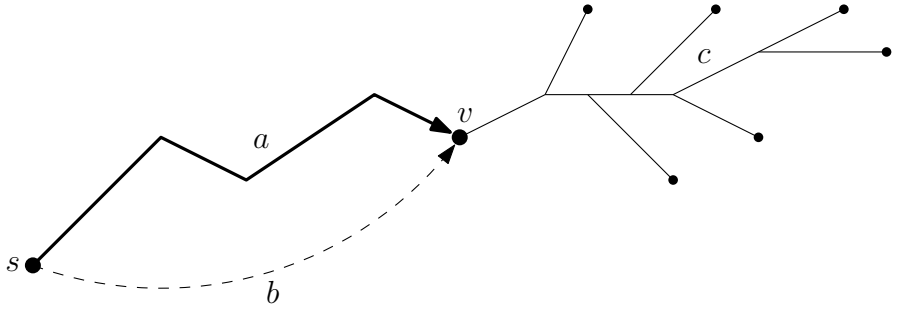


Figure 9.4.: Illustration of the tree-deletion pruning

different path b from s to v. Based on the label ℓ, we might already have built a tree of descendant labels c. If we proceed with the usual label-correcting algorithm, first the descendant labels of ℓ are pushed and later are dominated by the descendants of ℓ'. To avoid the unnecessary pushes of descendants of ℓ, we can delete the whole tree c after the label ℓ is deleted. Instead of traversing the queue of open labels to delete the tree labels, we mark labels as being deleted and actually delete them when they are popped from the queue. We call this pruning method *tree-deletion pruning (TD)*. We can employ this method in label-correcting algorithms using both, the label-selection and node-selection strategies. The tree-deletion pruning also plays an important role in the approximation algorithm which we introduce in Chapter 10.

This strategy is similar to the parent-checking heuristic for the single-objective shortest path problem in the work by Cherkassky, Goldberg, and Radzik [31]. But it is not so clear how the more lightweight methods can be implemented in the multiobjective setting.

We again conduct a computational experiment to investigate the performance gains when using the tree-deletion heuristic. We implemented TD into the label-correcting implementations with the LS and NS strategies. But since the NS strategy was considerably faster in the previous experiment and the picture did not change when using the tree-deletion heuristic, we only present the results of the label-correcting implementation using the NS strategy.

Following our philosophy to include as much detail about the implementations as possible, we give some more about how we integrated the tree-deletion part into the node-selection algorithm. The main implementation detail is how we store the successor labels. We decided to use a `std::list`. The reason for this is that the successor-lists are constructed empty and most of them remain empty for the whole process of the algorithm. Construction of empty `std::list`s is the cheapest operation among the creation of all other relevant data structures.

9.4.1. Question, Metrics and Experimental Setting

Our goal in this experiment is two-fold: First, we want to investigate if it happens often that labels are created which could have been avoided by using TD (Q1). Second, we want to know if our implementation does also improve the running time compared to the first NS implementation (Q2).

To answer Q1, we implement a number of hooks in our NS and LS code and instead of deleting a label in the course of TD, we mark them deleted and every time we push a label which is marked as being deleted, we count this event. This way we count the number of pushes which could be avoided by TD. We confine these experiments to the instances by Paixao and Santos [108] to get a general idea on the most important instance classes. We show box plots for all our instances to answer this question.

To answer question two, we implement a version of TD into our NS implementation. We see the running time differences by showing boxplots of the running times of the pure NS implementation and the implementation with TD. We measure the running times and present the differences in boxplots.

The experiments are again performed on an Intel Core i7-3770, 3.4 GHz and 16 GB of memory running Ubuntu Linux 12.04. The code is implemented in C++11 and compiled using LLVM 3.4 with compiler flag -O3.

9.4.2. Computational Study

Let us now turn to the results of the experiments.

Q1: Do unnecessary pushes happen and how often on the various instance classes?

The results can be seen in Figure 9.4. We observe that on these instances, especially the node-selection strategy tends to produce larger unnecessary trees than the label-selection strategy. The situation is different on the grid-graph instances, where both algorithms have a similar tendency to produce unnecessary trees.

Another observation is that when increasing the number of objectives, the number of unnecessary trees which could have been deleted decreases in the grid and random graph instances (see Figs. 9.4 b, d and f). This happens because when looking at instances with a large number of objectives and totally random objective values, most labels remain nondominated in the cleaning step. The node-selection implementation on complete-graph instances is an exception here, where the number of unnecessary pushes increases with the number of objectives. Which is similar to the situation of increasing number of nodes in the complete graph instances.

Hence, we expect that on instances where a large number of labels is dominated in the cleaning step the tree-deletion pruning is very useful. Especially the node-selection strategy, which is already faster than the label-selection strategy, should benefit from it.

objectives	running time (s) NS	NS-TD	$\lvert\mathcal{Y}_N\rvert$
BPA, LPA, L	4.04	**3.72**	377
BPA, EPL, L	13.41	**6.37**	1170
BPA, FW, L	1.13	**1.10**	280
BPA, SA, L	172.23	**76.80**	693
LPA, EPL, L	0.30	**0.24**	109
LPA, FW, L	6.59	**4.95**	634
LPA, SA, L	11.35	**7.49**	640
EPL, FW, L	1.65	**1.13**	428
EPL, SA, L	134.21	**52.82**	3767
FW, SA, L	40.98	**20.97**	1301
BPA, LPA, EPL, L	1549.16	**941.01**	11902
BPA, EPL, FW, L	—	**508.33**	13449
LPA, EPL, FW, L	336.38	**222.30**	8106

Table 9.2.: Running times and nondominated set sizes of the node-selection (NS) implementation and the node-selection implementation with tree-deletion (NS-TD) on the power grid optimization instances. In the penultimate instance, the NS implementation exceeded the memory limit of 16 GB.

Q2: Is the running time of the NS implementation faster using TD? How is the situation on the different instances?

The results of the comparison of the running times can be seen in Figures 9.5, 9.6 and Table 9.2. It can be seen on the real-world road networks that the tree-deletion pruning works very well and we can achieve a speed-up of up to 3.5 in comparison to the pure node-selection strategy.

On the instances from the power transmission line optimization set, we first see that we cannot optimize all objectives at once because of memory constraints which was 16 GB in these experiments. The combinations with at least three objectives which could be solved are shown in Table 9.2. But there is still one instance which could be solved by the NS implementation with TD but not with the NS implementation without TD with the given memory limit of 16 GB. We observe that the TD strategy again improves the performance of the node-selection strategy on these instances with a speed-up of up to 2.54.

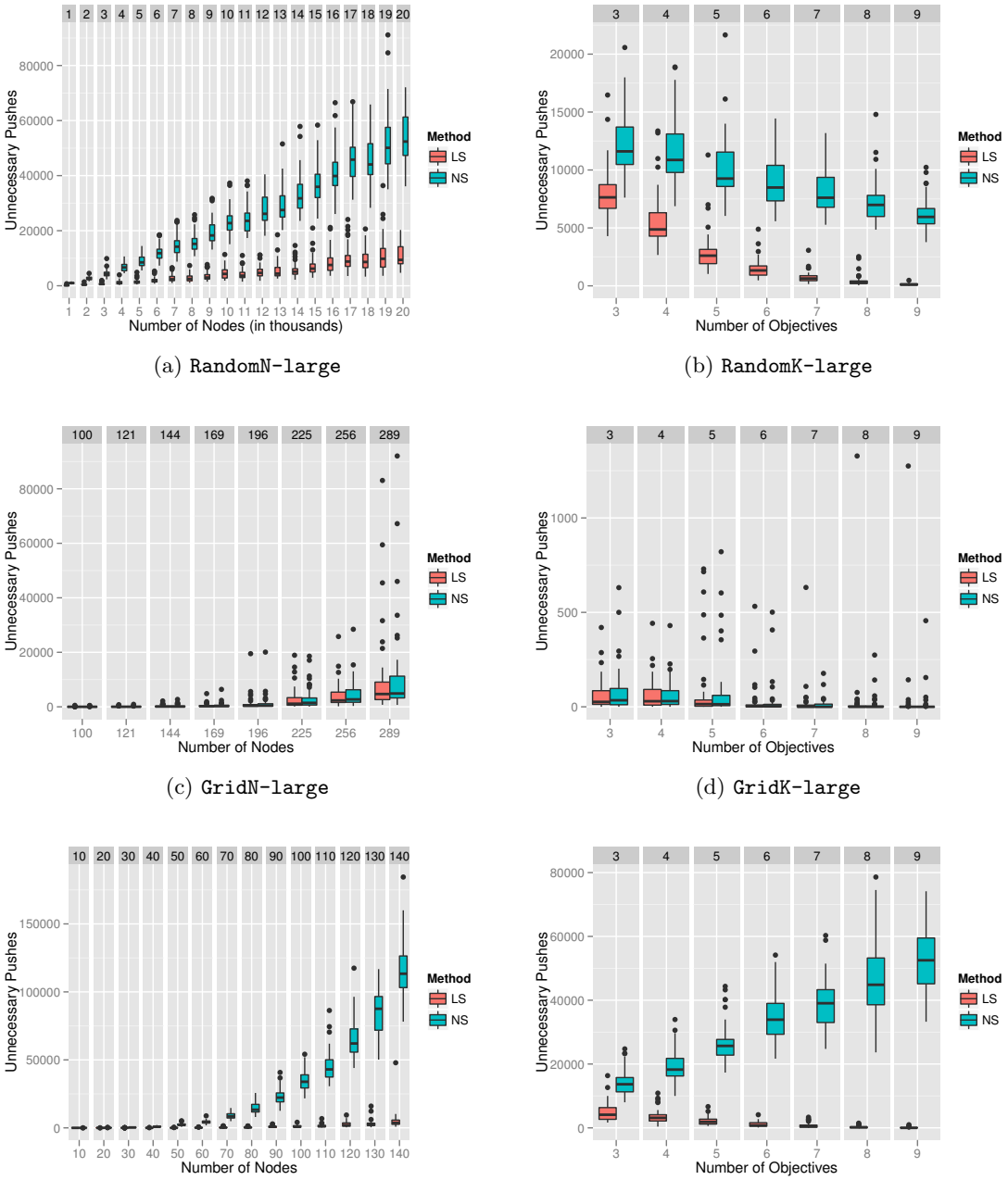
On the artificial benchmark instances however, the results are not so clear. On most instances of the artificial benchmark set, TD performs slightly worse than the pure node-selection strategy.

9.4.3. Discussion

It seems curious that the result is very positive on the real world instances, while TD is not effective on the artificial bechmark instances by Paixao and Santos [108].

This behavior can be explained by the large number of labels which are dominated in the road network and power grid instances. The sizes of the nondominated sets are small compared to the instances of the artificial test set. We can explain this behavior by the correlation between the objectives of these instances: In the artificial instances by Paixao and Santos [108], the weights are drawn independently at random and thus, no correlation exists between them. In the real-world instances, always some correlation can be observed: Bird preservation areas tend to be close to or inside landscape preservation areas, large settlements are usually not too far away from freeways and existing power grid lines.

Concerning the question if TD is a useful tool in power transmission line optimization, the answer is positive, besides the instances that could be solved with exact methods was comparably small.

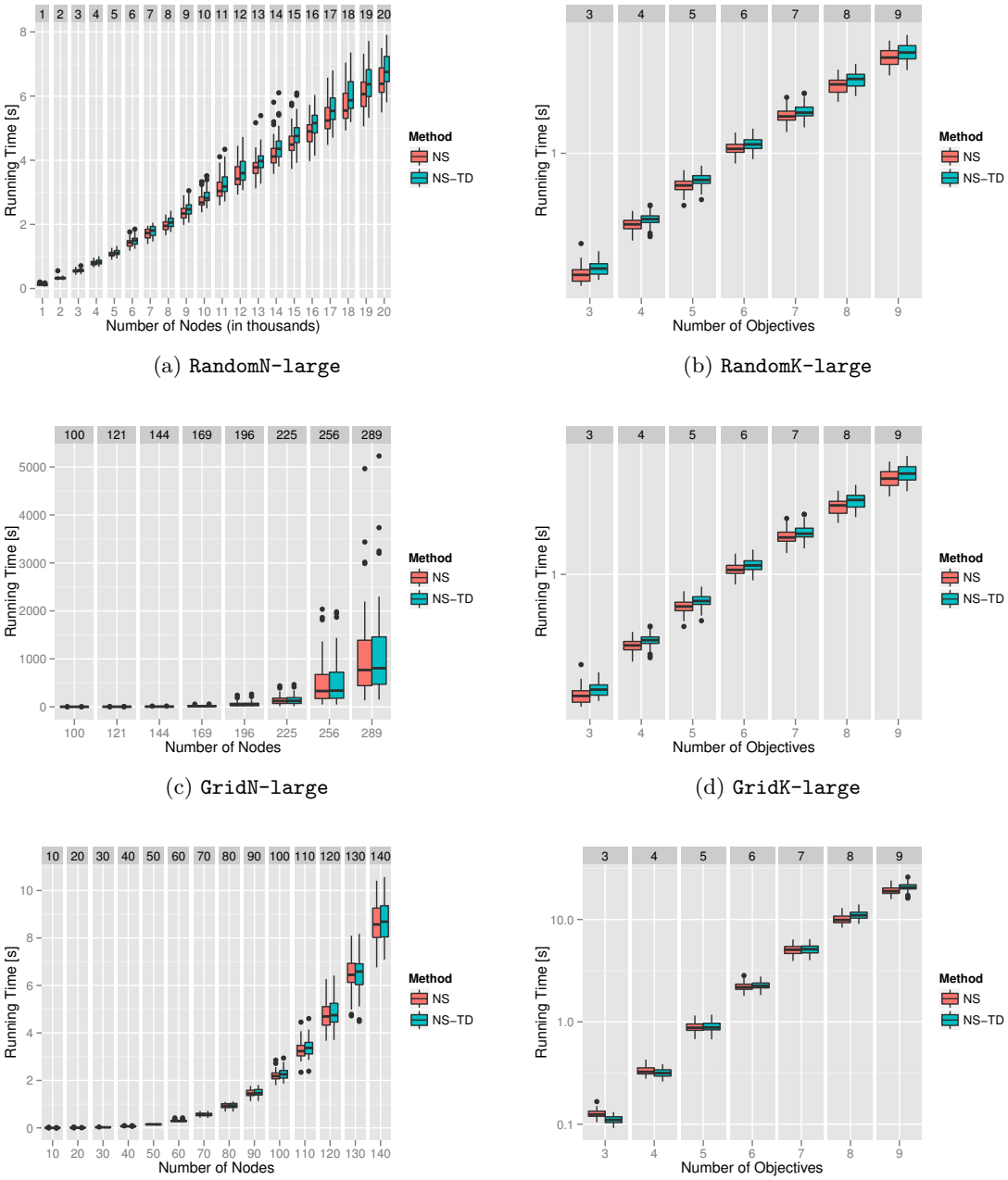


(a) `RandomN-large` (b) `RandomK-large`

(c) `GridN-large` (d) `GridK-large`

(e) `CompleteN-large` (f) `CompleteK-large`

Figure 9.5.: Measuring how many nodes have been touched which could have been deleted by tree-deletion pruning in the label-selection (LS) and node-selection (NS) strategies on the instances by Paixao and Santos [108]



(a) `RandomN-large`

(b) `RandomK-large`

(c) `GridN-large`

(d) `GridK-large`

(e) `CompleteN-large`

(f) `CompleteK-large`

Figure 9.6.: Comparison of the running times (in seconds) of the node-selection strategy with (NS-TD) and without (NS) tree-deletion pruning on the instances by Paixao and Santos [108]

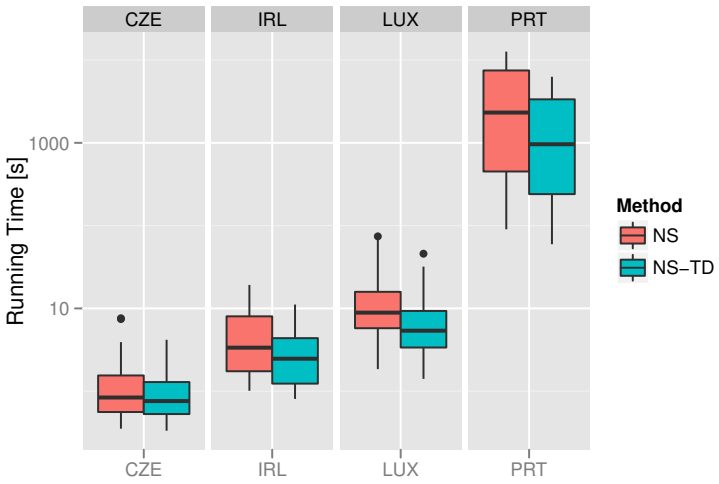


Figure 9.7.: Comparison of the running times (in seconds) of the node-selection strategy with (NS-TD) and without (NS) tree-deletion pruning on real road networks by PTV AG

10. Multiobjective Approximation

In the last chapter we have seen that exact methods turn out to be infeasible for our problem of identifying good power transmission lines. The next step thus is to investigate approximation algorithms. Let us first define what approximation in the context of multiobjective optimization means.

10.1. Models of Approximation

If a problem instance becomes very large, either in terms of number of variables or number of objectives, exact methods can be prohibitive. Instead, we want to have a good approximation of the nondominated set. But what makes a good approximation? Zitzler et al. [142] give a review and analyze various measures in multiobjective approximation. In summary they point out that the one measure that gives the best evaluation of an approximate Pareto set probably does not exist. There are measures which are better than others and some measures have advantages above others but are not entirely better. It depends heavily on the application and the circumstances involved in the solution process.

Keeping these considerations in mind, the following definitions arise from the study of approximation algorithm for multiobjective optimization problems in theoretical computer science as pursued by, e.g., Papadimitriou and Yannakakis [111] and many others. This definition of approximation is almost identical to the measure of the ε-indicator in the broader picture of multiobjective optimization. From a computer science perspective, this is a very appealing notion, since it resembles approximation in single-objective optimization. It also is a setting in which most progress could be made with respect to performance and quality guarantees.

For a vector $\boldsymbol{\varepsilon} \in \mathbb{R}^d_{\geq}$, we say a point $\boldsymbol{p}_1 \in \mathbb{R}^d_{\geq}$ $(\mathbf{1}+\boldsymbol{\varepsilon})$-*dominates* another point $\boldsymbol{p}_2 \in \mathbb{R}^d_{\geq}$ if $\boldsymbol{p}_1 \leq (\mathbf{1}+\boldsymbol{\varepsilon}) \otimes_{\mathrm{c}} \boldsymbol{p}_2$. Further we say, a set of points $M_1 \subseteq \mathbb{R}^d$ $(\mathbf{1}+\boldsymbol{\varepsilon})$-*covers* a set $M_2 \subseteq \mathbb{R}^d$ if for every point $\boldsymbol{p}_2 \in M_2$, there exists a point $\boldsymbol{p}_1 \in M_1$ such that $\boldsymbol{p}_1$ $(\mathbf{1}+\boldsymbol{\varepsilon})$-dominates $\boldsymbol{p}_2$. An example is shown in Fig. 10.1. Conclusively, for a MOCO problem with nondominated set $\mathcal{Y}_N$, a set $M \subseteq \mathbb{R}^d$ is an $(\mathbf{1}+\boldsymbol{\varepsilon})$-*Pareto set* if M $(\mathbf{1}+\boldsymbol{\varepsilon})$-covers $\mathcal{Y}_N$.

Definition 10.1 (Multiobjective $(\mathbf{1}+\boldsymbol{\varepsilon})$-approximation Algorithm)**.** For a MOCO problem O with nondominated set $\mathcal{Y}_N$, an algorithm $\mathcal{A}$ is an $(\mathbf{1}+\boldsymbol{\varepsilon})$-*approximation algorithm* for O if

1. $\mathcal{A}$ computes a set $M \subseteq \mathbb{Q}^d$ and M is an $(\mathbf{1}+\boldsymbol{\varepsilon})$-Pareto set for O, and
2. $\mathcal{A}$ runs in polynomial time in the instance size.

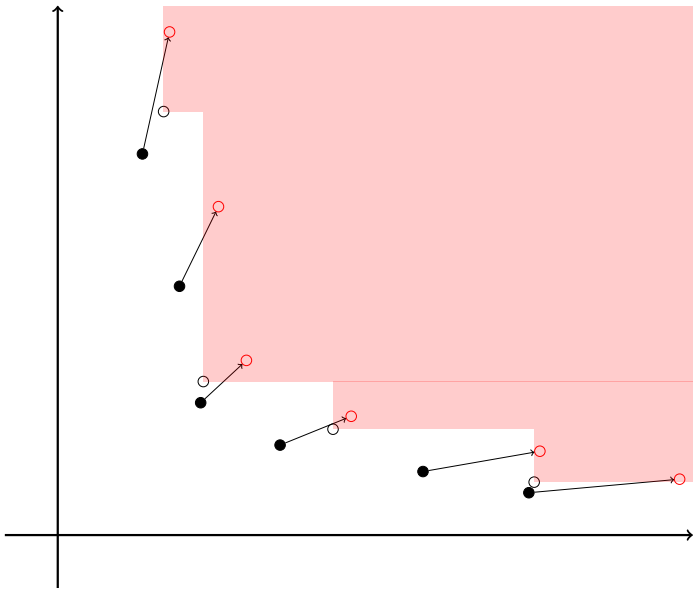


Figure 10.1.: An example for the definition of ε-cover: The set of hollow black points ε-covers the set of filled points for $\varepsilon = \frac{3}{2} \cdot \mathbf{1}$. The red circles are the black points scaled by $\frac{3}{2}$.

The following definition is analogous to the definition of FPTASes in single-objective optimization and most of the literature and all of the following is concerned with this multiobjective equivalent.

Definition 10.2 (Multiobjective FPTAS (Fully Polynomial-Time Approximation Scheme))**.** For a MOCO problem O with nondominated set $\mathcal{Y}_N$ we say an algorithm $\mathcal{A}$ with additional input of $\varepsilon \in \mathbb{Q}^d_>$ is an *FPTAS* for O, if for every ε it holds that $\mathcal{A}$ is an $(\mathbf{1} + \varepsilon)$-approximation algorithm for O with running time bounded polynomially in the input size and ε^{-1}.

10.2. Literature Review

In this section we give a short overview on the existing literature of approximation algorithms for the multiobjective shortest path problem with performance and quality guarantees in the given model.

10.2.1. Early works by Hansen and Warburton

The first approximation algorithm for the MOSP problem was proposed by P. Hansen in 1979 [74]. The algorithm is similar to the classical rounding algorithm for the knapsack problem, and operates only on biobjective instances. In this algorithm, all coefficients of one objective function are scaled down and the floor of each of these

numbers is taken to obtain only integer numbers. We obtain a new instance by keeping the old instance and only replacing one objective function by its scaled version. Then, we can use a pseudopolynomial labeling algorithm like the one given in Hansen [74] to obtain the nondominated set of the scaled instances. As proven by Hansen, this yields an $\boldsymbol{\varepsilon}$-Pareto set for the biobjective shortest path problem.

A more sophisticated algorithm, capable of approximating the general MOSP problem was proposed by A. Warburton in 1987 [138]. The basic idea is the following: If we have an upper bound U on the cost of the paths in every objective, we can divide the objective space into exponentially growing cells, where we set $B := \log_2 U$ and cell $C_{\boldsymbol{j}}$ for $\boldsymbol{j} \in (0 \cup [B])^{d-1}$ is the set $\{\boldsymbol{x} \in \mathbb{R}^{d-1} \mid 2^{\boldsymbol{j}_i} < \boldsymbol{x}_i \leq 2^{\boldsymbol{j}_i+\mathbf{1}}, i \in [d-1]\}$. For each cell we run a pseudopolynomial MOSP algorithm with costs of the first $d-1$ objective functions scaled depending on the cell to obtain nondominated vectors for this cell, again similar to the FPTAS for the Knapsack problem. An example of such a pseudopolynomial algorithm is the algorithm by Martins [98].

The key insight to the running time is that the number of cells is bounded by the logarithm of the cost of the most expensive path per objective in the graph. And thus only a polynomial number of cells need to be considered and the pseudopolynomial algorithm always runs in polynomial time due to the scaling.

10.2.2. A General Approximation Approach

In their work and many subsequent papers, Papadimitriou and Yannakakis showed in their paper from 2000 [111] that it is possible to generalize the ideas by Warburton to a very general class of multiobjective optimization problems. First, they prove that for every MOP and every $\varepsilon > 0$, there exists an $(\mathbf{1} + \varepsilon \cdot \mathbf{1})$-Pareto set of size polynomial in the instance size and ε^{-1} (but exponential in the number of objectives d). The basic idea is to decompose the objective space into slightly different cells than the idea by Warburton (cf. Fig. 10.2). For a given $\boldsymbol{\varepsilon} = \varepsilon \cdot \mathbf{1}$, we set $B := \log_{1+\varepsilon} U$ and the cell $\boldsymbol{j}$ for $\boldsymbol{j} \in (\{0\} \cup B)^{d-1}$ is the set $C_{\boldsymbol{j}} := \{x \in \mathbb{R} \mid (1+\boldsymbol{\varepsilon}_i)^{\boldsymbol{j}_i - 1} < x_i \leq (1+\boldsymbol{\varepsilon}_i)^{\boldsymbol{j}_i}, i \in [d-1]\}$. If we have a set of points M such that there is a point in every cell $(\mathbf{1}+\boldsymbol{\varepsilon})$-covering every point in this cell then M is an $(\mathbf{1}+\boldsymbol{\varepsilon})$-cover of the nondominated set and M has a size of $\mathcal{O}(B^{d-1})$ which is polynomial in the instance size and ε^{-1} for each fixed d.

Further, they prove that an FPTAS exists if we can solve a so-called *gap problem* in polynomial time. The gap problem is almost the problem A. Warburton solved to get the nondominated set for a cell, but in the gap problem only one point needs to be found. This idea is a generalization of the idea of Warburton. Moreover, they also show that the converse direction of this implication holds, i.e., if the gap problem cannot be solved in polynomial time, then there is no FPTAS for the MOP at hand.

This theory was further refined to find succinct $\boldsymbol{\varepsilon}$-Pareto sets. The existence result above and also the algorithms do not guarantee that the sets output are anywhere near a small size, albeit their polynomiality. Vassilvitskii and Yannakakis [137] proved that in the case of two objectives an $\boldsymbol{\varepsilon}$-Pareto set of size at most three times the smallest such set can be found in polynomial time and that smaller sets cannot be found in polynomial time. For the case of three and more objectives the same authors show

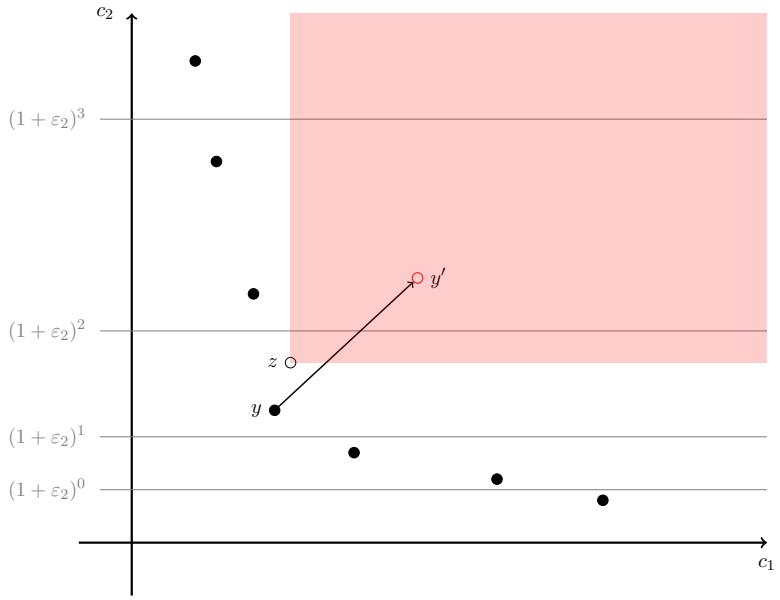


Figure 10.2.: The cell structure introduced by Papadimitriou and Yannakakis [111]. In this figure, $\boldsymbol{\varepsilon} = \mathbf{1}$. If we scale point y by $\mathbf{1} + \boldsymbol{\varepsilon}$, we get a point in a cell at the boundary of the same cell or in another cell with a larger index.

that no $\boldsymbol{\varepsilon}$-Pareto set of size within a constant factor from the smallest such set can be found in polynomial time. Other relaxations of this model also exists by Diakonikolas [45] and Diakonikolas and Yannakakis [46, 47].

10.2.3. The FPTAS by Tsaggouris and Zaroliagis

The most recent approximation algorithm for the MOSP problem was proposed by Tsaggouris and Zaroliagis [134] in 2009. Instead of filling the cells of the decomposition given by Papadimitriou and Yannakakis each cell at a time, the algorithm by Tsaggouris and Zaroliagis executes only one labeling algorithm and fills the cells while the labeling algorithm runs. Thus, for each node the algorithm maintains a table of cells as a $(d-1)$ dimensional array. In each cell only the cost of the last objective function is stored and always filled with the path with the least value of the last objective.

The ith position in the table of the cell for the label $\ell = (\boldsymbol{v}, u, \hat{\ell})$ of a given path is computed by the function $\mathrm{pos}^{\boldsymbol{r}} \colon \mathbb{Q}_{>}^{d-1} \to \mathbb{N}^{d-1}$ depending on $\boldsymbol{r} \in \mathbb{Q}_{\geq}^{d-1}$:

$$\mathrm{pos}_i^{\boldsymbol{r}}(\boldsymbol{v}) = \begin{cases} \lfloor \log_{1+\boldsymbol{r}_i} \frac{\boldsymbol{v}_i}{c_i^{\min}} \rfloor & \text{if } \boldsymbol{v}_i \geq 1 \text{ and } c_i^{\min} > 1 \\ \lfloor \log_{1+\boldsymbol{r}_i} \boldsymbol{v}_i \rfloor & \text{if } \boldsymbol{v}_i \geq 1 \text{ and } c_i^{\min} \leq 1 \\ 0 & \text{else} \end{cases}$$

where $c_i^{\min} = \min\{c_i(a) \mid a \in A\}$ for $i \in [d]$.

The labeling algorithm follows a traditional Bellman-Ford algorithm also described in the book by Ehrgott [55]. Albeit, Bellman-Ford labeling strategies are not well tested in the literature of MOSP in practice, but the proof of correctness relies heavily on the fact that it is a Bellman-Ford procedure.

10.3. A General Approximation Scheme

We now describe how we can derive a general approximation scheme from the work by Tsaggouris and Zaroliagis [134]. Similar to them, we consider only one run of a multiobjective labeling algorithm. But we do not limit ourselves to a Bellman-Ford method and apply the idea to general multiobjective labeling algorithms. This is especially interesting in practice, since there is a lot of work in the literature about speeding up label-setting and label-correcting implementations.

Consider Algorithm 5 as a generic example of a labeling algorithm—it is our model-algorithm in this section. Notice that this algorithm is a generalization of Listings 3 and 4. The generalization happens in Line 5, where a general selection rule is employed. This way, the algorithm can also be seen as a generalization of the Bellman-Ford algorithm as used in [134].

Listing 5 A Generic MOSP Labeling Algorithm

Require: Graph $G = (V, A)$, nodes $s, t \in V$, objective function $c\colon A \to \mathbb{Q}^d$ and a vector $r \in \mathbb{Q}^d_{\geq}$

Ensure: List R of pairs (p, y) for all $y \in c(\mathcal{P}_{s,t})$ and some $p \in \mathcal{P}_{s,t}$ such that $c(p) = y$

1: $L_u \leftarrow \emptyset$ for all $u \in V \backslash \{s\}$
2: Initialize label ℓ at node s
3: $L_s \leftarrow \{\ell\}$
4: **while** there are still labels left **do**
5: Select a node v and a subset of labels L of $\mathcal{L}_v$
6: **for each** label $\ell \in L$ **do**
7: **for each** $(u, w) \in A$ **do**
8: Push ℓ along (u, w) and add the new label ℓ' to L_w
9: Clean L_w
10: **if** ℓ' is nondominated in L_w **then**
11: Q.push(ℓ')
12: Reconstruct paths for each label in L_t and output path/vector pairs

Departing from Algorithm 5, we extend each label $\ell = (\boldsymbol{v}, u, \hat{\ell})$ by the position in the table given by the decomposition of Papadimitriou and Yannakakis and get new labels $\ell = (\boldsymbol{v}, \boldsymbol{p}, u, \hat{\ell})$. If we push a label ℓ along an arc $a = (u, w)$, we obtain a new label $\tilde{\ell} = (\boldsymbol{v} + c(a), \mathrm{pos}(\boldsymbol{v} + c(a)), w, \ell)$. Here, we extend the function pos to $\mathrm{pos} : \mathbb{Q}^d_{>} \to \mathbb{N}^d$ to cover all objective functions.

When adding a new label at a node u, we have to check whether the new label is dominated and do the clean up step in Line 9 of Listing 5. We perform almost the

same dominance check as in Listings 3 and 4, but use the index of the Papadimitriou and Yannakakis decomposition and the component-wise $<$ relation instead of the $\leq$ relation. This way, labels with cost vectors in a cell which have at least one same or smaller index are kept. Only labels with an index strictly larger in every objective are discarded. If a label with the same Papadimitriou-Yannakakis-index is pushed, we always keep the old label and discard the new one. The goal of this step is to speed up the algorithm considerably in practice while still guaranteeing the same approximation ratio.

Now, we analyze this algorithm. To run the algorithm, we need to specify the vector $\boldsymbol{r}$ for the $\text{pos}^{\boldsymbol{r}}$-function. Clearly, $\boldsymbol{r}$ needs to be chosen depending on the given $\boldsymbol{\varepsilon}$. In our analysis, we change this viewpoint at first. Given a vector $\boldsymbol{r} \in \mathbb{Q}^d_{\geq}$, we prove that we can arrive at the same $\boldsymbol{\varepsilon}$ as in the proof by Tsaggouris and Zaroliagis [134] no matter what selection procedure we use. The running time then depends on the selection routine.

Lemma 10.1 (Loss on a Path: Upper Bound)**.** *Given an instance of the MOSP problem and a vector $\boldsymbol{r} \in \mathbb{Q}^d_{\geq}$. Let $p^* = (s = v_1, \ldots, v_{l+1} = t)$ be a Pareto-optimal path and $e_i \coloneqq (v_i, v_{i+1})$ for $i \in [\overline{l}]$. Independent of how the selection routine is implemented, the algorithm finds a path p with cost*

$$c(p)_i \leq \sum_{k=1}^{l} c(e_k)_i (1 + \boldsymbol{r}_i)^{l-k+1}.$$

for every $i \in [d]$.

Proof. Let a graph $G = (V, A)$ be given with objective function $c\colon A \to \mathbb{Q}^d$ and source node s and target node t. Let p^* be given as above.

Now let us assume the algorithm ran on the instance and let the selection routine be arbitrary. This defines the sets L_v in the end of the algorithm, which depend on the selection routine used. If the algorithm finds a label at t corresponding to p^*, we are done. So let us assume p^* is not found, then there is a node $v \coloneqq v_i \in p^* = (v_1, \ldots, v_{l+1})$ with the smallest index i such that there is no label in L_v corresponding to the subpath $(s, \ldots, v)$ of p^*. Let $p^*_v = (s, \ldots, v)$ be the subpath of p^* from s to v.

Claim 1. There is a label $\ell = (\boldsymbol{v}, \boldsymbol{p}, v, \hat{\ell}) \in L_v$ with $\boldsymbol{p} = \text{pos}^{\boldsymbol{r}}(c(p^*_v))$.

Proof. Assume there is no such label, then there is a label $\ell = (\boldsymbol{v}, \boldsymbol{p}, v, \hat{\ell}) \in L_v$ with $\boldsymbol{p} < \text{pos}^{\boldsymbol{r}}(c(p^*_v))$ which dominated a label induced by p^*, otherwise there existed a label for p^*_v. But then $\boldsymbol{v} \preceq c(p^*)$ and that cannot be true since every subpath of p^* is also Pareto-optimal. □

Now since we know that there is a label $\ell = (\boldsymbol{v}, \boldsymbol{p}, v, \hat{\ell}) \in L_v$ with $\boldsymbol{p} = \text{pos}^{\boldsymbol{r}}(c(p^*_v))$, we know that $\boldsymbol{v} \leq (1 + \boldsymbol{r})c(p^*_v)$, because of the definition of $\text{pos}^{\boldsymbol{r}}$. Let $\hat{p_v}$ be the path corresponding to the label ℓ.

We can now construct a new path taking the arcs of $\hat{p_v}$ from s to v and the arcs of p^* from v to t, let this path be $\hat{p}$. The path $\hat{p}$ is not necessarily simple, but if it is not, we can construct a new path from s to t from $\hat{p}$ by eliminating all cycles on the path and we see that $c(\hat{p})$ is an upper bound on its cost and the path uses fewer arcs of p^*.

Now we can iterate the above considerations with $\hat{p}$ taking the place of p^*. Each time we show that there must be a label along this path found by the labeling algorithm independent of the selection method. We end up with a path $\hat{p}$ from s to t with at most k iterations with a label $\ell = (\boldsymbol{v}, \boldsymbol{p}, v, \hat{\ell})$ at node t. In each iteration we lose a factor of $(1 + \boldsymbol{r}_i)$ times the cost of the iteration before in every objective. Thus $c(\hat{p})_i \leq ((c(e_1)_i(1 + \boldsymbol{r}_i) + c(e_2)_i)(1 + \boldsymbol{r}_i) + c(e_3)_i)(1 + \boldsymbol{r}_i) + ...$

Claim 2. $c(\hat{p})_i \leq \sum_{k=1}^{l} c(e_k)_i(1 + \boldsymbol{r}_i)^{l-k+1}$ for all $i \in [d]$

Proof. We prove the claim by induction over the length l of the Pareto-optimal path p^*. If $l = 1$, then $c(\hat{p})_i \leq c(e_1)_i(1 + \boldsymbol{r}_i)$ since we can only lose a factor of $(1 + \boldsymbol{r}_i)$ once. Now let's assume the claim is true for $l' - 1$, i.e., we already lost $\sum_{k=1}^{l'-1} c(e_k)_i(1 + \boldsymbol{r}_i)^{(l'-1)-k+1}$ in each objective. If the path was one edge longer, e.g., having the additional edge $e_{l'}$, we lose $(\mathbf{1} + \boldsymbol{\varepsilon})$ for this edge and for all costs we had before, i.e.,

$$
\begin{aligned}
c(\hat{p})_i &\leq (\sum_{k=1}^{l'-1} c(e_k)_i(1 + \boldsymbol{r}_i)^{(l'-1)-k+1} + c(e_{l'})_i)(1 + \boldsymbol{r}_i) \\
&= (c(e_1)_i(1 + \boldsymbol{r}_i)^{l'-1} + c(e_2)_i(1 + \boldsymbol{r}_i)^{(l'-1)-1} + \cdots + c(e_{l'-1})_i(1 + \boldsymbol{r}_i) + c(e_{l'})_i)(1 + \boldsymbol{r}_i) \\
&= c(e_1)_i(1 + \boldsymbol{r}_i)^{l'} + c(e_2)_i(1 + \boldsymbol{r}_i)^{l'-1} + \cdots + c(e_{l'-1})_i(1 + \boldsymbol{r}_i)^2 + c(e_{l'})_i(1 + \boldsymbol{r}_i) \\
&= \sum_{k=1}^{l'} c(e_k)_i(1 + \boldsymbol{r}_i)^{l'-k+1}
\end{aligned}
$$

□

This way we have seen that the algorithm constructs a label at t with cost at most $\sum_{k=1}^{l} c(e_k)_i(1 + \boldsymbol{r}_i)^{l-k+1}$ for all $i \in [d]$ which proves the lemma. □

Accordingly, to compute an $\boldsymbol{r}$ from a given $\boldsymbol{\varepsilon}$, we can safely set $\boldsymbol{r}_i := \boldsymbol{\varepsilon}_i^{\frac{1}{n-1}}$. We thus use that no path can have more than $n - 1$ edges which is a very rough bound. If we have more information about maximum path length, we can adapt this computation of $\boldsymbol{r}$ accordingly.

Let us now discuss the running time of the approximation algorithm. In contrast to the algorithm by Tsaggouris and Zaroliagis, we only store labels for paths that we really need. In the worst case, however, we need to store as many labels as the number of cells in the tables used in the algorithm by Tsaggouris and Zaroliagis. Hence in the worst case, we do not gain anything. Even worse, we have to perform a dominance check for each new label we produce. In practice, this is not so bad assuming that the worst-case amount of labels is not created. In the worst-case, however, we have to perform a dominance check even on the worst-case amount of labels. In the algorithm

by Tsaggouris and Zaroliagis, a dominance check is not even required in the end because dominated labels are allowed by the definition of $\boldsymbol{\varepsilon}$-Pareto sets.

For the sake of completeness, we give the running-time of the algorithm if we use a Bellmann-Ford selection scheme and for a label-setting selection scheme. A label-correcting scheme does not yield good worst-case running times because, in the worst-case, many labels can be pushed which are dominated later. But first, we reproduce a lemma by Tsaggouris and Zaroliagis [134], to show a bound on the number of labels we get. In the following, let $n := |V|, m := |E|, c^{\max} := \max\{c_i^{\max} \mid i \in [d]\}$ and $\varepsilon := \max\{\boldsymbol{\varepsilon}_i \mid i \in [d]\}$.

Lemma 10.2 (Tsaggouris and Zaroliagis [134])**.** *For a given MOSP instance and $\boldsymbol{\varepsilon} \geq \mathbf{0}$, we have $|L_v| \in \mathcal{O}((\frac{n \log(nc^{\max})}{\varepsilon})^d)$ for every $v \in V$ independently of the selection routine and at any given point in the algorithm.*

Proof. Let $\boldsymbol{r}_i := (1 + \boldsymbol{\varepsilon}_i)^{\frac{1}{n-1}}$. A path's cost does not exceed $nc_i^{\max}$ in objective $i \in [d]$. Accordingly, we have at most $N := \prod_{i=1}^{d} \log_{\boldsymbol{r}_i}(nc_i^{\max})$ many labels at each node because it is an upper bound on the number of values the pos-function can attain. Using that

$$\log_{\boldsymbol{r}_i} x = \frac{\log x}{\log \boldsymbol{r}_i}, \text{ for } x > 0,$$

and because $\log(1 + x) = \Theta(x)$ for small x, we have

$$\begin{aligned} N &= \mathcal{O}(\prod_{i=1}^{d} \frac{\log nc_i^{\max}}{\log \boldsymbol{r}_i}) \\ &= \mathcal{O}(\prod_{i=1}^{d} \frac{\log nc_i^{\max}}{\log(1 + \boldsymbol{\varepsilon}_i)^{\frac{1}{n-1}}}) \\ &= \mathcal{O}(\prod_{i=1}^{d} \frac{(n-1) \log nc_i^{\max}}{\boldsymbol{\varepsilon}_i}) \\ &= \mathcal{O}(n^d \prod_{i=1}^{d} \frac{\log nc_i^{\max}}{\boldsymbol{\varepsilon}_i}) \end{aligned}$$

□

Now, we can investigate the running time using a Bellmann-Ford strategy.

Proposition 10.3. *When a Bellmann-Ford selection routine is employed, the running time of the algorithm is*

$$\mathcal{O}(n^{d+1} m (\frac{\log(nc^{\max})}{\varepsilon})^d \log^{d-2}(n(\frac{\log(nc^{\max})}{\varepsilon})))$$

for each fixed number of objectives.

Proof. We have $n-1$ iterations and in each iteration, we inspect all edges and push at most all $N := \mathcal{O}(n^d \prod_{i=1}^{d} \frac{1}{\varepsilon_i} \log(nc_i^{\max}))$ (cf. Lemma 10.2) labels at every node. For each such push we remove all dominated labels at the target node. The worst-case running time for the clean-up step in a set of k vectors with t components is bounded by $\mathcal{O}(k \log^{t-2} k)$ [90]. Consequently, the running time is $\mathcal{O}(nmN \log^{d-2} N))$. □

In Chapter 9 we introduced the label-setting strategy by Martins [98]. In contrast to other label-correcting strategies, we can give a polynomial upper bound for the approximation algorithm with this strategy.

Proposition 10.4. *When using the lexicographic selection routine from the label-setting algorithm by Martins [98], the running time of the approximation algorithm can be bounded by*

$$\mathcal{O}(n^{2d+2}(\frac{\log(nc^{\max})}{\varepsilon})^{2d} \log^{d-2}(n\frac{\log(nc^{\max})}{\varepsilon}))$$

for each fixed number of objectives.

Proof. According to Lemma 10.2, at most $N := \mathcal{O}(n^d \prod_{i=1}^{d} \frac{1}{\varepsilon_i} \log(nc_i^{\max}))$ labels are stored per node. Thus in total, we store at most nN labels at all nodes. If the priority queue is implemented as a Fibonacci heap (cf. Fredman and Tarjan [64]), we get the following results: Because of the label-setting property, we extract at most nN labels from the queue. For each such label, we insert at most n new labels, giving a total running time for all inserts of $\mathcal{O}(n^2 N)$. Consequently, the queue contains at most $n^2 N$ elements and the extract-min operation is performed at most nN times, resulting in a running time of $\mathcal{O}(nN \log nN)$ for all extract-min operations.

For each label we push across an edge, we perform a dominance check. The dominance check on k labels with d objectives takes time at most $\mathcal{O}(k \log^{d-2} k)$ [90]. We create $n^2 N$ labels at most and we have N labels per node, so the running time for all dominance checks is $\mathcal{O}(n^2 N^2 \log^{d-2} N)$. Hence, the running time totals to $\mathcal{O}(n^2 N^2 \log^{d-2} N)$. □

For comparison, the running time of the fastest as yet known algorithm by Tsaggouris and Zaroliagis [134] is

$$\mathcal{O}(n^d m(\frac{\log(nc^{\max})}{\varepsilon})^{d-1}).$$

The theoretical running times do not look too promising, but we show in the next section, that the worst-case is not the regular case.

10.4. Comparison of Practical Running Time and Ratio

We now investigate the performance in practice. Coming back to our motivational problem, our goal is to find an approximation that works well in practice, especially with more than only four or five objectives. While the analysis again suggests, that

the running time is upper bounded exponentially in the number of objectives, we conjecture that the running time does not usually reach this upper bound.

Further, the analysis and especially Lemma 10.1 is very defensive: We assume that we lose a factor of $(1 + \boldsymbol{r}_i)$ in every objective $i \in [d]$ on every edge of a path of maximum length. But we suppose that this is highly improbable in practice. Thus, assuming to set $\boldsymbol{\varepsilon}(n)_i = \boldsymbol{r}_i^{\frac{1}{n-1}}$ might as well be too defensive. We test another possible function $\boldsymbol{\varepsilon}(n)$, namely $\boldsymbol{\varepsilon}'(n)_i = \boldsymbol{r}_i^{\frac{1}{\log(n-1)}}$, and see how much faster the algorithm is and how much we lose in quality on the instances tested. But, we note that this way we lose the quality guarantees.

10.4.1. The Implementation

Departing from our studies in Chapter 9, we implemented a label-correcting version of our approximation algorithm. We call this implementation LC Approx. The same reasonings as in Section 9.3 apply and we also use the naive cleaning procedure due to the larger number of objectives.

Additionally, we implemented the tree-deletion heuristic from Section 9.4 for two reasons: First, especially on the transmission grid instances we did see a performance gain. Second, there is a subtle memory problem compared to the original algorithm by Tsaggouris and Zaroliagis: Suppose we have a label ℓ and the predecessor label of ℓ is deleted. In the approximation algorithm, it can happen that ℓ is never deleted by other labels and persists until the algorithm terminates. This is problematic in two ways: The labels unnecessarily occupy memory. And a more severe reason is that we run into problems when computing the paths in the end of the algorithm: A label whose predecessor got deleted does not get its predecessor pointer set into a valid state. Thus, if that label is not deleted, the reconstruction of the paths represented by the labels at t at some point discovers ℓ and tries following its predecessor pointer.

As an aside: In the algorithm by Tsaggouris and Zaroliagis, the predecessor is always valid, because it points into a table. We cannot guarantee that the label at the destination cell is the original predecessor, but we can check whether the costs match. If they do not, we can safely discard this path. If they do, we can try to further construct the paths and if it contains a cycle we can again discard it and thus always end up with a feasible path.

10.4.2. State-of-the-Art Algorithms and Implementations

To the best of our knowledge there are no implementations of the algorithms by Warburton [138] and Tsaggouris and Zaroliagis [134] or any approximation algorithm for MOSP available. Further, there are no studies investigating the empirical running times of these algorithms. So we are faced with a bleak picture concerning the running times and quality in practice and we implemented both of the above methods ourselves.

The method by Warburton is a special case of the scheme by Papadimitriou and Yannakakis [111] in the following sense: To solve the gap problem, a pseudopolynomial

algorithm for the constrained shortest path (CSP) problem needs to be used. State-of-the-art pseudopolynomial algorithms for the CSP problem are labeling algorithms which solve the multiobjective extension of the CSP instance and can discard labels which exceed the bounds (cf., e.g., Dumitrescu and Boland [49]). An example of such an algorithm is the label-setting algorithm by Martins [98]. Moreover, while there is a considerable literature on the CSP with one resource constraint, not much is known for the case of an arbitrary number of resource constraints. The grid in the algorithm by Warburton is always the size of the grid of the algorithm by Papadimitriou and Yannakakis with $\varepsilon = 1$. For smaller ε, the size of the grid increases in the algorithm of Papadimitriou and Yannakakis, but stays the same size in the algorithm by Warburton. To be faster in practice, we use our node-selection label-correcting algorithm to solve the scaled subproblems and to further accelerate the implementation, we permute the objectives such that the objective i with the largest $c_i^{\max}/c_i^{\min}$ is last, where $c_i^{\max}$ is defined analogously to $c_i^{\min}$.

In the implementation of the algorithm by Tsaggouris and Zaroliagis, we use two tables per node, one for the current labels and one for the labels of the last iteration. We switch the semantics of each table in each iteration. The Bellmann-Ford algorithm frame is implemented in a straight forward way.

In our preexperiments, we see that the implementation of the algorithm by Tsaggouris and Zaroliagis is very slow even on instances with 3 objectives. This comes to no surprise, since the table for every node which consists of a number of entries which is exponential in the number of objectives needs to be checked in every iteration. Hence, we decided to exclude this implementation from our further study.

10.4.3. Questions, Metrics and Experimental Setting

First, we want to compare the running-times of the LC Approx implementation and the Warburton implementation and see which one is faster while guaranteeing the same quality guarantees. We formulate this as question one (Q1). To test the implementations, we use $\boldsymbol{\varepsilon} \coloneqq \alpha \cdot \mathbf{1}$ for $\alpha \in \{0.1, 0.25, 0.5, 1\}$.

It is not only interesting which algorithm is faster, but also how the quality develops over the instances. We thus differentiate between the guaranteed quality and the empirical quality. One peculiarity can be observed here: While it is easy to decide if a given set of points is an $(\mathbf{1} + \boldsymbol{\varepsilon})$-Pareto set for a given instance and $\boldsymbol{\varepsilon}$, the vector $\boldsymbol{\varepsilon}$ for which a given set of points is an $(\mathbf{1} + \boldsymbol{\varepsilon})$-Pareto set for a given instance is not unique. Thus, if we want to evaluate a set of points for a given instance we need a metric of how good the approximation is. Since in our experiments we set $\boldsymbol{\varepsilon}_1 = \cdots = \boldsymbol{\varepsilon}_d$, it is a reasonable choice to measure the quality of an approximation by the smallest α such that the set of points is a $(\mathbf{1} + \alpha \cdot \mathbf{1})$-Pareto set. So our second question (Q2) is, which algorithm gives the better empirical performance under the same performance guarantee under the above measure.

Following our observation that it is highly improbable that the worst-case assumptions of Lemma 10.1 apply, we also test the LC Approx algorithm with a smaller $\boldsymbol{r}$ for a given $\boldsymbol{\varepsilon}$. More specifically, we chose $\boldsymbol{r}_i \coloneqq \boldsymbol{\varepsilon}_i^{\frac{1}{\log(n-1)}}$ for every $i \in [d]$ and assume to

get a much faster empirical running time, but we also lose the performance guarantee. We call this implementation the LC Approx (log) implementation, although the only thing that changes is the way how it computes $\boldsymbol{r}$. Our third question (Q3) thus is, how much to we gain in speed and more importantly, do we really lose much in the performance and how does it develop over the instances?

The experiments were performed on an Intel Core i7-3770, 3.4 GHz and 16 GB of memory running Ubuntu Linux 16.04. The algorithms are implemented in C++11 and the code is compiled using LLVM 3.4 with compiler flag -O3. For all experiments there is a memory limit of 16 GB and a time limit of one hour.

10.4.4. Computational Study

We now describe the results of the computational study and group the results by our research questions. As the results are very similar for the various choices of α, we present the results for $\alpha = 1$.

Q1: Which implementation is faster?

On the transmission grid instances, we observe a mixed result. See Table 10.1 for the details. The Warburton implementation is faster on almost all instances with three objectives and gives roughly the same quality as the LC Approx implementation. On the instances with four objective functions, the Warburton implementation is not able to terminate on even one out of four instances in the given time limit. The LC Approx implementation, however, is able to complete three out of four instances in less than 8min. The hardest instance with objectives BPA, LPA, FW and L could only be finished by the LC Approx (log) implementation in the given limits. This instance was also not solved in the tree-deletion experiments by either of the exact methods (cf. Section 9.4) due to the memory limit.

On the real-world road instances by the PTV AG (cf. Figure 10.3) the picture is very much the same as for the transmission grid instances with three objectives: The median running time of the Warburton implementation is better than the median running time of the LC Approx implementation. But in this instance set we see that there exist heavy outliers on the side of the Warburton implementation. While the median on all three instance sets is less than 3s, the farthest outlier has a running time of about 634s, 1689s and 2705s on the CZE, IRL, and LUX instances, respectively. The LC Approx implementation on the other hand has a maximum over all instance at 146.2s and does not have any point which is farther away from the box than 1.5 times the interquartile range while still having the median at about 16.5s. One has to bear in mind that the PTV instances only have three objective functions.

The results on the artificial benchmark instances by Paixao and Santos [108] with increasing number of objectives is depicted in Figure 10.4. A general picture for all instances from the set by Paixao and Santos emerges: The curve of the running times of the Warburton implementation is always above the curve of the running times of the LC Approx implementation which again is above the curve of the running times of

	LC Approx		LC Approx (log)		Warburton	
objectives	time [s]	ratio	time [s]	ratio	time [s]	ratio
BPA, LPA, L	4.11	1.0026	0.04	1.4200	1.38	1.0051
BPA, EPL, L	5.96	1.0051	0.03	1.3390	3.57	1.0664
BPA, FW, L	1.24	1.0024	0.03	1.2084	0.47	1.0000
BPA, SA, L	47.72	1.0026	0.04	1.2907	2.78	1.0019
LPA, EPL, L	0.50	1.0031	0.02	1.2120	0.10	1.0014
LPA, FW, L	5.18	1.0026	0.04	1.4531	12.92	1.0011
LPA, SA, L	6.36	1.0035	0.04	1.5203	1.49	1.0007
EPL, FW, L	1.06	1.0054	0.03	1.3097	5.21	1.2302
FW, SA, L	10.61	1.0034	0.04	1.3013	16.55	1.0033
BPA, LPA, EPL, L	480.17	1.0054	0.07	1.4200	—	—
BPA, EPL, FW, L	88.05	1.0059	0.04	1.4867	—	—
LPA, EPL, FW, L	74.42	1.0059	0.06	1.4531	—	—
BPA, LPA, FW, L	—	—	0.13	1.5763	—	—

Table 10.1.: Running times (in seconds) and empirical ratios of the implementations of the label-correcting approximation algorithm (LC Approx), the label-correcting approximation with logarithmic ratio (LC Approx (log)) and the method by A. Warburton (Warburton) on the power transmission grid optimization instances. In the cases where the running time is marked with a '—', one of the limits was exceeded. These are the results for $\alpha = 1$.

the LC Approx (log) implementation. We can clearly see that the implementation of the Warburton algorithm scales badly with the number of objectives. The time limit is already reached on instances with 5 objectives on complete and random instances or 6 objectives on grid instances. Our LC Approximation implementation is able to solve every instance with up to 9 objectives in less than 40s whereas the implementation with logarithmic scaling of $\boldsymbol{r}$ solves every instance in less than 6s. The difference in the LC Approx and LC Approx (log) implementations are larger than the boxplots with logarithmic ordinate axis suggests. It appears as if it is constant but the difference actually increases with increasing number of objectives.

The results on the instance set by Paixao and Santos [108] with increasing number of nodes and fixed six objectives can be found in Figures 10.5 – 10.7. We expected the Warburton implementation to perform a lot better when the number of objectives is fixed but it turns out that the implementation can only solve the most basic instances. On the `GridN-large` and `RandomN-large` instances, it can not even solve the smallest instance with 6 objectives in the given limit. Nevertheless, we show the running time of the Warburton implementation on the smallest instance with 100 and 1000 nodes—which by far exceeds the running time limit of 1 hour—to show this relation. On the `CompleteN-large` instances, we can see a bit of the scaling with increasing number of nodes. The LC Approx implementation is far superior and the LC Approx

(log) implementation is again even faster.

We see a comparison of the running times and ratios on the instance set by Paixao and Santos with $\alpha \in \{0.1, 0.25, 0.5, 1\}$ [1] in Figures 10.8 and 10.9. We see that the running time of the LC Approx implementation is almost invariant to the setting of α in this range and the reason for this can be seen in the measured ratio. Already for $\alpha = 1$, it often hits the exact nondominated set and the measured ratio is 1.0. By selecting smaller α, we do not get significantly more labels and thus the algorithm performs almost the same computations. For the Warburton implementation this picture is mixed: On the `CompleteK-large` instance set, larger ε result in much faster running times. Whereas not much changes on the `CompleteN-large` and `GridK-large` instances. On the `GridN-large` and `RandomN-large` instances, the running times exceeded the 1 hour limit even on the smallest instance. The running time of the LC Approx (log) implementation, however, is very sensitive to the ε chosen.

Q2: Which implementation gives better empirical approximation quality?

On the real-world road networks and on the transmission grid instances the LC Approx and Warburton implementation have about the same empirical quality. The LC Approx implementation seems to be more robust and stable on these instances.

This impression is also visible on the artificial networks by Paixao and Santos. The LC implementation is a bit ahead in quality compared to the Warburton implementation especially on the `GridK-large` instances. But the differences are not significant,

[1] As a reminder: $\boldsymbol{\varepsilon} := \alpha \mathbf{1}$

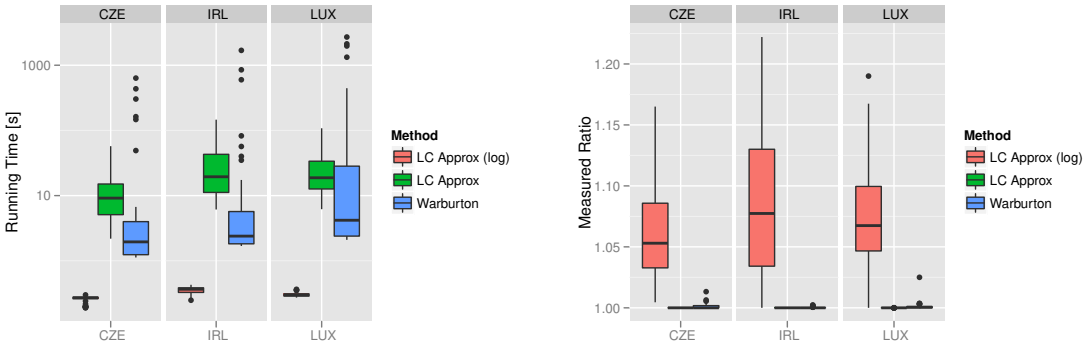
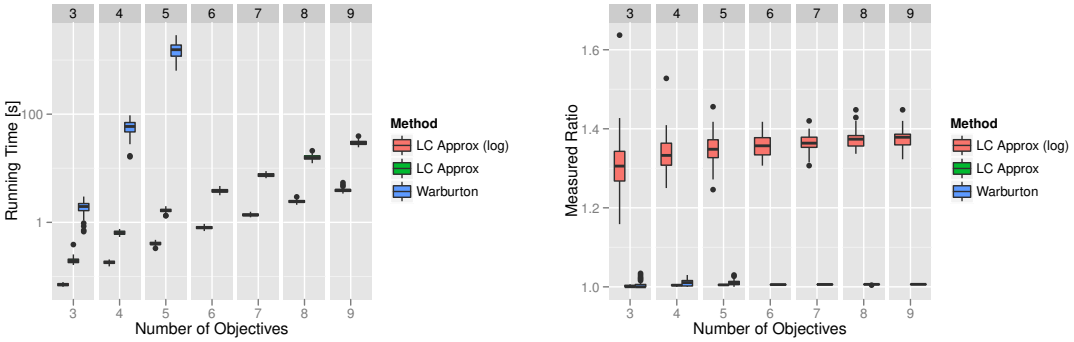
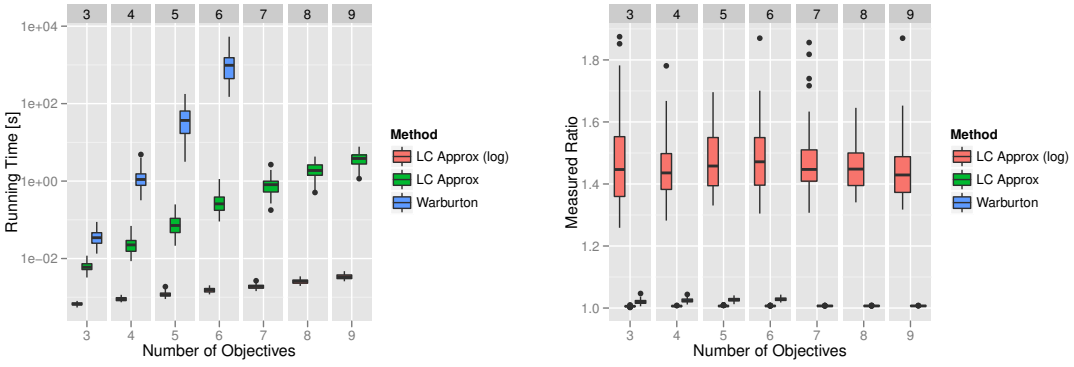


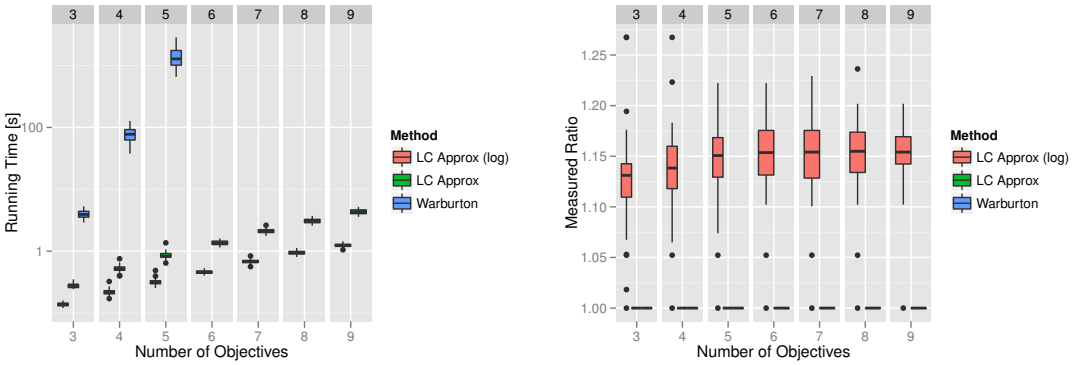
Figure 10.3.: Running times and measured approximation ratios of the label-correcting approximation algorithm (LC Approx), the label-correcting approximation with logarithmic ratio (LC Approx (log)) and the method by A. Warburton (Warburton) on the real-world road network by PTV AG. The ordinate axis in the running time plot is logarithmic. These are the results for $\alpha = 1$.



(a) `CompleteK-large`



(b) `GridK-large`



(c) `RandomK-large`

Figure 10.4.: Comparison of the running times (in seconds) and measured approximation ratios of the label-correcting approximation algorithm (LC Approx), the label-correcting approximation with logarithmic ratio (LC Approx (log)) and the method by A. Warburton (Warburton) on the instances by Paixao and Santos [108]. These are the results for $\alpha = 1$. The ordinate axes in the running time plots are logarithmic. The Warburton implementation exceeded the time limits on instances with at least 6, 7, and 6 objectives, respectively. Thus, if there are only two boxes visible, those are the boxes from the LC Approx (log) and LC Approx implementation (in this order).

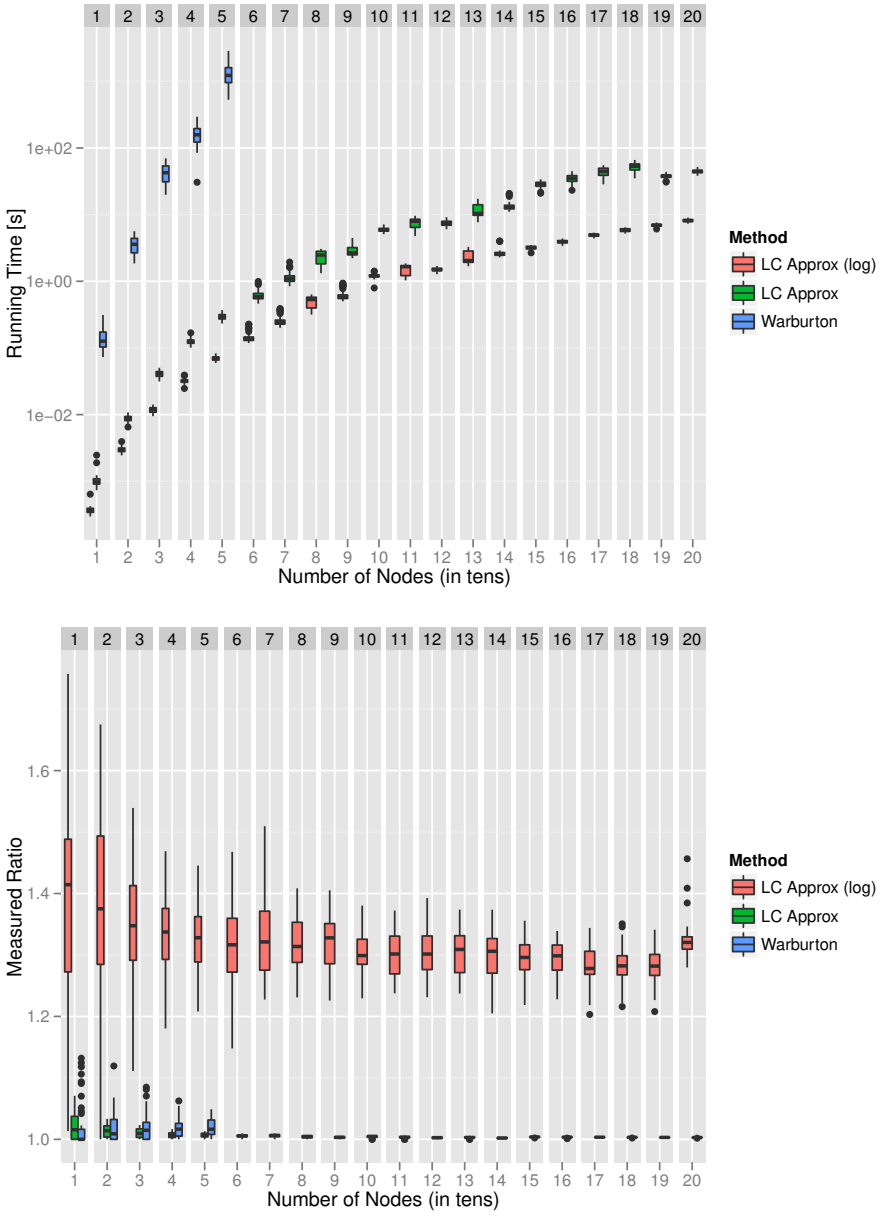


Figure 10.5.: Comparison of the running times (in seconds) and measured approximation ratios of the label-correcting approximation algorithm (LC Approx), the label-correcting approximation with logarithmic ratio (LC Approx (log)) and the method by A. Warburton (Warburton) on the `CompleteN-large` instances by Paixao and Santos [108]. These are the results for $\alpha = 1$. The ordinate axis in the running time plot is logarithmic. The Warburton implementation was only able to solve instances with up to 50 nodes in the given time limit. Thus, if there are only two boxes visible, those are the boxes from the LC Approx (log) and LC Approx implementation (in this order).

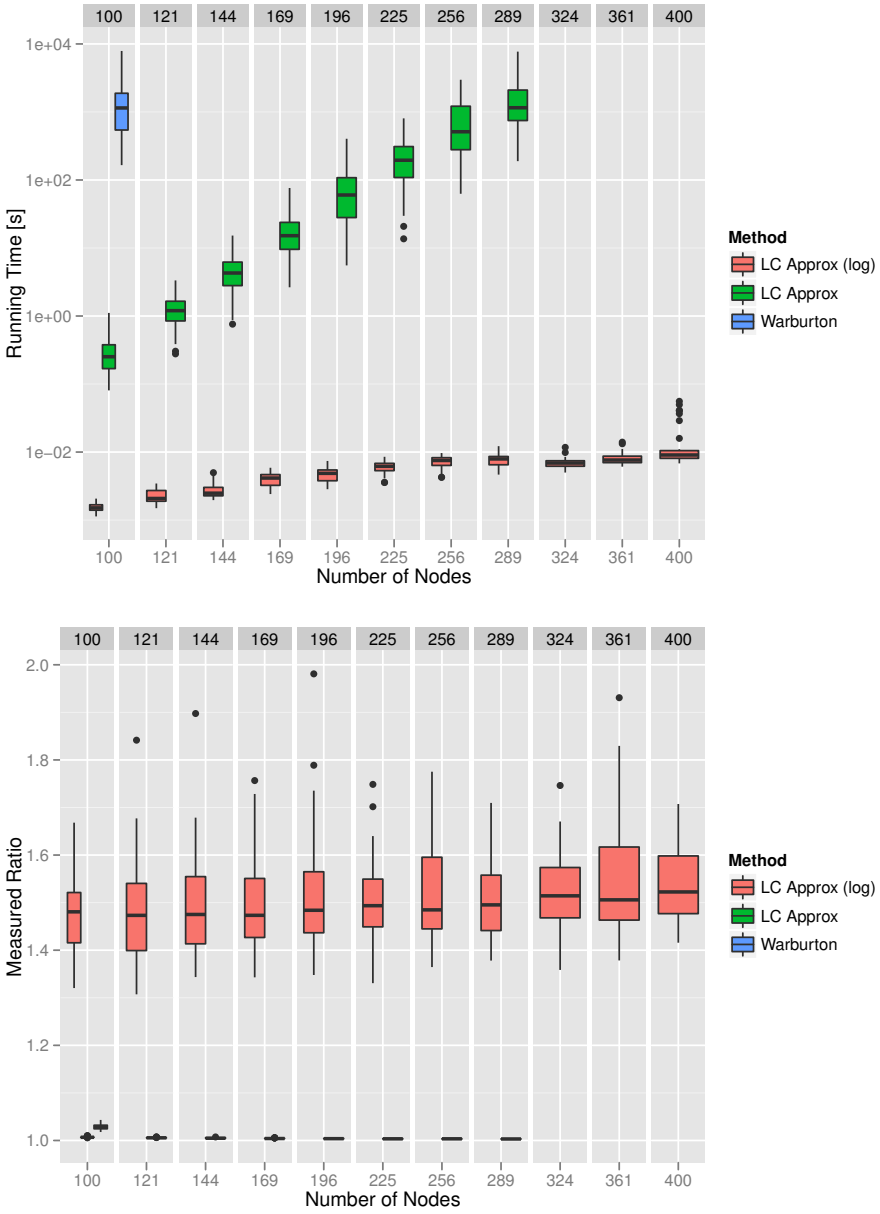


Figure 10.6.: Comparison of the running times (in seconds) and measured approximation ratios of the label-correcting approximation algorithm (LC Approx), the label-correcting approximation with logarithmic ratio (LC Approx (log)) and the method by A. Warburton (Warburton) on the `GridN-large` instances by Paixao and Santos [108]. These are the results for $\alpha = 1$. The ordinate axis in the running time plot is logarithmic. The Warburton implementation exceeded the time limit on all instances, we include the result for $n = 100$ for reference. Thus, if there are only two boxes visible, those are the boxes from the LC Approx (log) and LC Approx implementation (in this order).

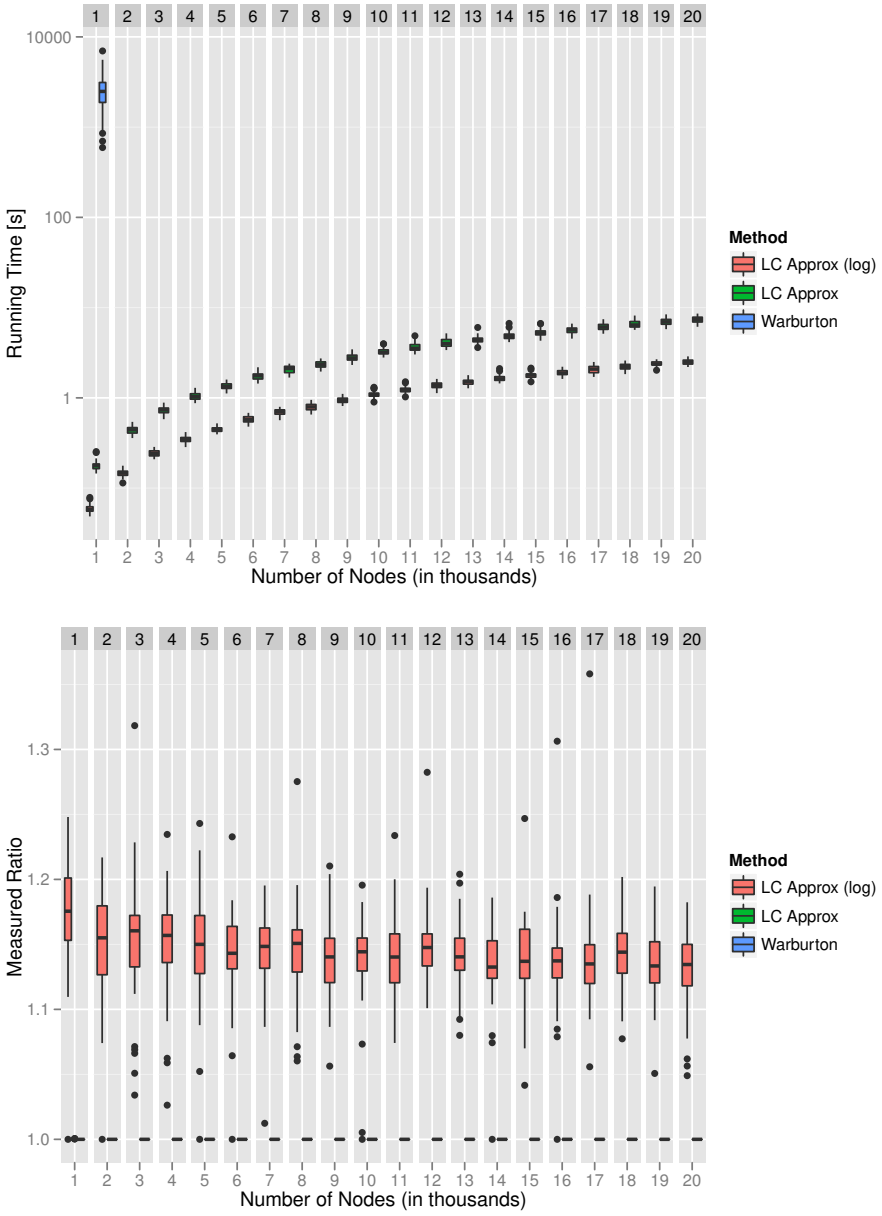


Figure 10.7.: Comparison of the running times (in seconds) and measured approximation ratios of the label-correcting approximation algorithm (LC Approx), the label-correcting approximation with logarithmic ratio (LC Approx (log)) and the method by A. Warburton (Warburton) on the `RandomN-large` instances by Paixao and Santos [108]. These are the results for $\alpha = 1$. The ordinate axis in the running time plot is logarithmic. The Warburton implementation exceeded the time limit on all instances, we include the result for $n = 1000$ for reference. Thus, if there are only two boxes visible, those are the boxes from the LC Approx (log) and LC Approx implementation (in this order).

especially since the measured quality of both implementations are far better than the guarantee.

Q3: Is a larger choice of r and losing the performance guarantee reasonable?

On the transmission grid instances the results are very promising. The instance with objectives BPA, LPA, FW and L could not be finished by any of the methods seen until this point of this thesis in the given limits. We were only able to compute the exact nondominated set, because we had access to a computer with more than 128GB of memory. The LC-log implementation took only 0.13s and a bit under 1 GB of memory. At the same time the goal ratio of $2 \cdot \mathbf{1}$ was still not exceeded, the actual ratio was about 1.57.

On the road networks the running times with at most 0.43s is also very impressive while at the same time never exceeding the empirical ratio of 1.23 and thus being far away from the goal of $2 \cdot \mathbf{1}$.

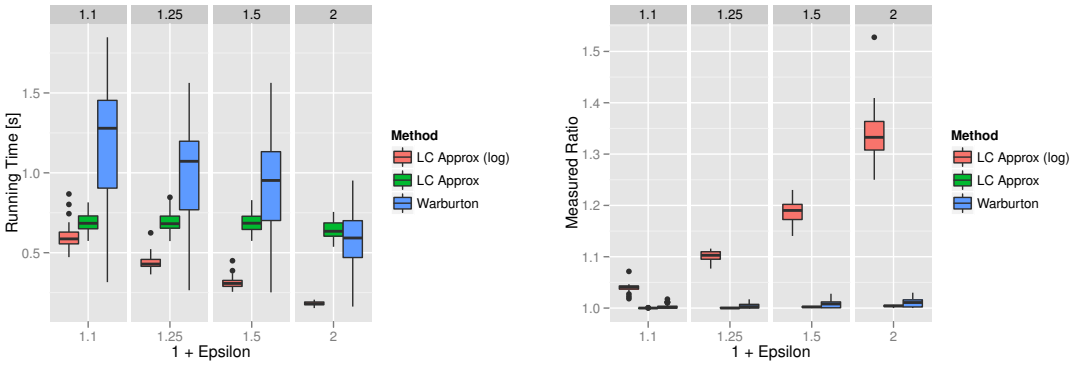
Comparing the ratios on the artificial benchmark instances with increasing number of objectives by Paixao and Santos, we can observe the same results. The worst measured approximation ratio is 1.875 on an instance from the `GridK-large` set. An explanation for this happening on the grid instances is that the paths are forced to have a minimum length of $2(n-1)$ for an $n \times n$ grid to move from one corner to the opposing corner. While the path length in complete or random graphs are presumably shorter. It occurs that the measured ratio does not generally worsen with an increasing number of objectives, which is what the analysis also suggests.

More interesting in this context is how the measured ratio changes when the number of nodes is varied and thus the expected path length. But we get the same picture on the complete and random graphs: On the complete graph instances the measured ratio gets even better and stabilized with growing number of nodes. But on the grid graph instances, the performance is still satisfying, but there is one outlier almost worse than the target ratio of 2 on an instance with $14 \cdot 14$ nodes.

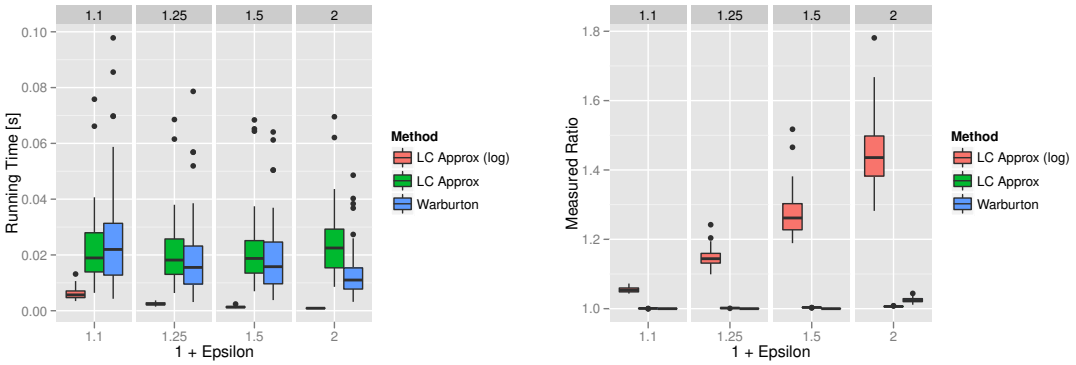
When changing α, there is one instance in the `GridK-large` family which for a given $\alpha = 0.5$ exceeds this bound. But this is the only test in which this happened. The influence of the input ε on the measured ratio is as expected. There is no clear picture visible that suggests that the quotient of measured ratio over input ratio gets worse with increasing or decreasing input ε.

10.4.5. Discussion

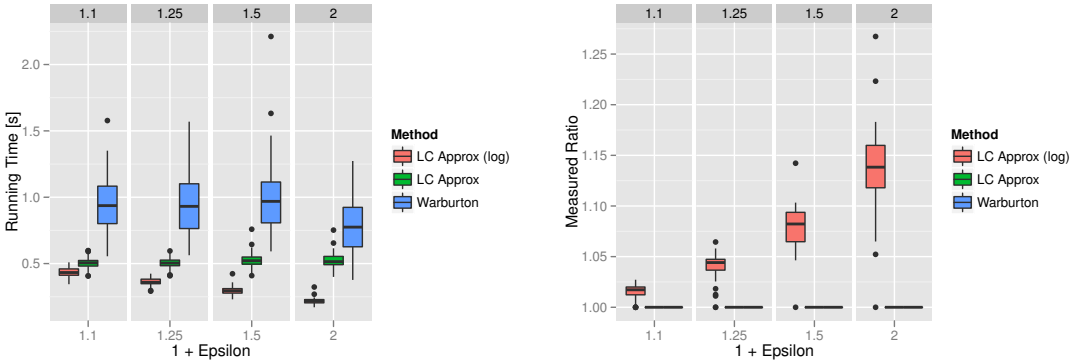
It seems that the LC Approx implementation did what it was designed for: Giving good quality guarantees even on instances with a larger number of objectives. Starting at about 4 objective functions, the LC Approx implementation dominates the running time of the Warburton implementation on every instance, usually in orders of magnitude. And at the same time it gives the same theoretical performance guarantees and also is more robust in the measured performance. The performance of the Warburton



(a) `CompleteK-large`

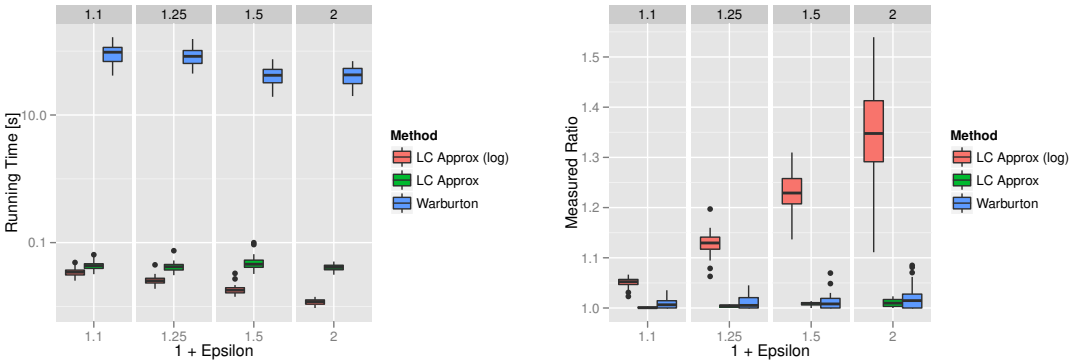


(b) `GridK-large`

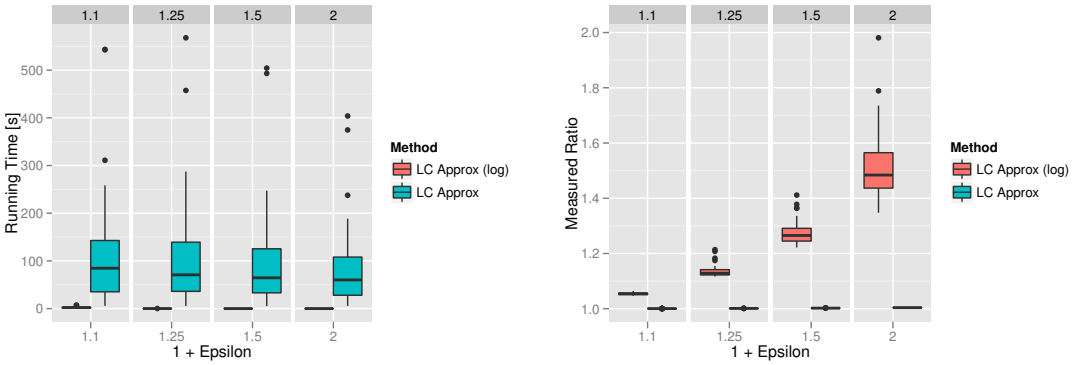


(c) `RandomK-large`

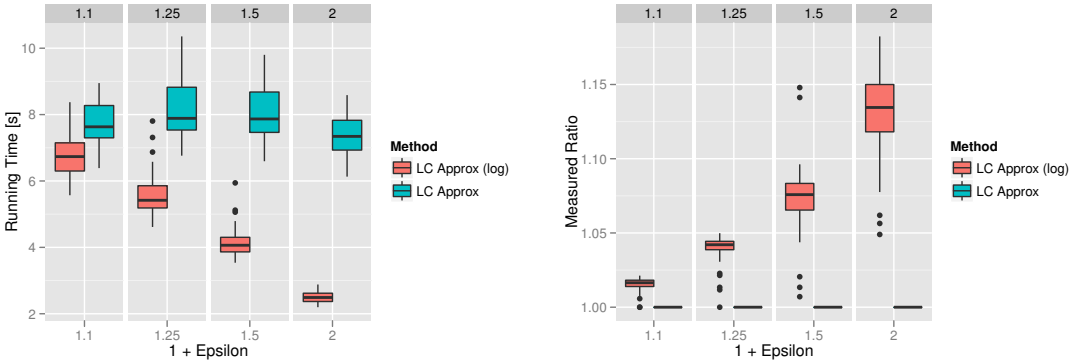
Figure 10.8.: Comparison of the running times (in seconds) and measured approximation ratios of the label-correcting approximation algorithm (LC Approx), the label-correcting approximation with logarithmic ratio (LC Approx (log)) and the method by A. Warburton (Warburton) on the instances by Paixao and Santos [108] varying the input ε.



(a) `CompleteN-large`



(b) `GridN-large`



(c) `RandomN-large`

Figure 10.9.: Comparison of the running times (in seconds) and measured approximation ratios of the label-correcting approximation algorithm (LC Approx), the label-correcting approximation with logarithmic ratio (LC Approx (log)) and the method by A. Warburton (Warburton) on the instances by Paixao and Santos [108] varying the input ε. The ordinate axis in the running time plot for the `CompleteN-large` instances is logarithmic.

implementation was also rather disappointing since it was slower than the exact labeling algorithms by orders of magnitude on instances with 4 or more objectives. The LC Approx implementation on the other hand is never much slower than an equivalent label-correcting implementation since it is almost the same algorithm: If we set $\boldsymbol{r}$ close to 0, we only have to deal with the superfluous creation and bookkeeping of the pos-vectors. The difference in the domination check is almost unnoticeable since it is not very probable that two labels share the same cost in one component.

Concerning the LC Approx (log) implementation, the results are very interesting. It seems to be a reasonable choice to raise the values of $\boldsymbol{r}$ to speed the running time up and lower the memory requirements. The goal was exceeded only once on a grid graph instance with $\alpha = 0.5$. But concerning the generalizability of these findings, we also have to say that there might be a graph size from which on the Approx LC (log) implementation performs worse and exceeds the goal. But up to a certain graph size it can still be used, if the exact ratio is not critical. In a specific application, a reasonable choice of $\boldsymbol{r}$ should always be tested beforehand.

11. Conclusion

Returning to the introduction of this part, our goal was to find an algorithm which works well on MOSP instances with more than just a few objectives. To achieve this, in Chapter 9, we review multiobjective labeling algorithms which are the most promising algorithmic methodologies for the MOSP problem and present some preliminary experimentation. We show that the common belief that the label selection strategy is superior to the node selection strategy is most probably wrong and show that even the exact opposite can be assumed. We also present a speed-up heuristic for label-correcting algorithms which speeds-up the running time considerably especially on the real-world instances from the literature and the power transmission line optimization problem.

But since the instances of the real power transmission optimization problem are a lot larger than what can be solved efficiently with exact algorithms, we turn to approximation methods in Chapter 10. We present a multiobjective FPTAS with the special property that what is known about practically efficient labeling algorithms can be incorporated into implementations of the algorithm. In our extensive computational experiments we show that our implementation of the approximation algorithm is far superior to the other state-of-the-art FPTAS from the literature. Especially, it scales much better with the number of objectives and even instances with up to 9 objectives can be approximated in reasonable time.

We did the whole analysis of the labeling algorithms under the assumption that label-correction methods are superior to label-setting methods when the number of objectives grow. This assumption is based on the thesis in the studies by Guerriero and Musmanno [71] and Paixao and Santos [108]. As the source code is not available, we cannot verify this thesis directly. Nevertheless, the algorithmic scheme we developed in Chapter 10 is also applicable to a label-setting algorithm as the one by Martins [98].

Open Problems

A systematic study with published source code is needed to state reliable theses on the performance of algorithms for the multiobjective shortest path problem. At this point, it is definitively not clear which algorithm to implement in which situation. The demonstration of the inconsistencies regarding label or node selection strategies is only a small area in this science landscape; the question of when to use label-setting and when to use label-correcting algorithms is a larger one.

In general, there does exist a substantial amount of research for approximate methods (with performance guarantees) for multiobjective optimization problems. But

they are implemented only seldom or are hopeless to implement, particularly because of a dependence on the number of objectives. It is an interesting field of research to find approximation algorithms which are useful in practice. The results from Part I of this thesis about MOLP might be of help in this regard.

Bibliography

[1] Ackermann, H., A. Newman, H. Röglin, and B. Vöcking (2005). "Decision Making Based on Approximate and Smoothed Pareto Curves." In: *Algorithms and Computation. ISAAC 2005*. Ed. by X. Deng and D. Du. Lecture Notes in Computer Science 3827. Springer, Berlin, Heidelberg.

[2] Aggarwal, A. and J. S. Vitter (1988). "The input/output complexity of sorting and related problems." In: *Communications of the ACM* 31(9), pp. 1116–1127.

[3] Aissi, H., A. R. Mahjoub, S. T. McCormick, and M. Queyranne (2014). "A Strongly Polynomial Time Algorithm for Multicriteria Global Minimum Cuts." In: *Integer Programming and Combinatorial Optimization. IPCO 2014*. Ed. by J. Lee and J. Vygen. Lecture Notes in Computer Science 8494. Springer, Cham, pp. 25–36.

[4] Aneja, Y. P. and K. P. K. Nair (1979). "Bicriteria Transportation Problem." In: *Management Science* 25(1), pp. 73–78.

[5] Armand, P. (1993). "Finding all maximal efficient faces in multiobjective linear programming." In: *Mathematical Programming* 61(1–3), pp. 357–375.

[6] Armand, P. and C. Malivert (1991). "Determination of the efficient set in multiobjective linear programming." In: *Journal of Optimization Theory and Applications* 70(3), pp. 467–489.

[7] Armon, A. and U. Zwick (2006). "Multicriteria Global Minimum Cuts." In: *Algorithmica* 46(1), pp. 15–26.

[8] Arora, S. and B. Barak (2009). *Computational Complexity – A Modern Approach*. Cambridge University Press, Cambridge, England.

[9] Avis, D. and K. Fukuda (1992). "A pivoting algorithm for convex hulls and vertex enumeration of arrangements of polyhedra." In: *Discrete and Computational Geometry* 8(1), pp. 295–313.

[10] Bauer, R. and D. Delling (2009). "SHARC: Fast and robust unidirectional routing." In: *Journal of Experimental Algorithmics* 14(4), pp. 2.4–2.29.

[11] Benson, H. P. (1998a). "An Outer Approximation Algorithm for Generating All Efficient Extreme Points in the Outcome Set of a Multiple Objective Linear Programming Problem." In: *Journal of Global Optimization* 13(1), pp. 1–24.

[12] Benson, H. P. (Apr. 1998b). "Further Analysis of an Outcome Set-Based Algorithm for Multiple-Objective Linear Programming." In: *Journal of Optimization Theory and Applications* 97(1), pp. 1–10.

[13] Benson, H. P. (1998c). "Hybrid Approach for Solving Multiple-Objective Linear Programs in Outcome Space." In: *Journal of Optimization Theory and Applications* 98(1), pp. 17–35.

[14] Benson, H. P. and E. Sun (2000). "Outcome Space Partition of the Weight Set in Multiobjective Linear Programming." In: *Journal of Optimization Theory and Applications* 105(1), pp. 17–36.

[15] Bertsekas, D. P., F. Guerriero, and R. Musmanno (1996). "Parallel asynchronous label-correcting methods for shortest paths." In: *Journal of Optimization Theory and Applications* 88(2), pp. 297–320.

[16] Bogh, K. S., I. Assent, and M. Magnani (2013). "Efficient GPU-based skyline computation." In: *International Workshop on Data Management on New Hardware. DaMoN 2013*. ACM, New York.

[17] Bökler, F. (2017). "The Multiobjective Shortest Path Problem is NP-hard, or is It?" In: *Evolutionary Multi-Criterion Optimization. EMO 2017*. Ed. by H. Trautmann, G. Rudolph, K. Klamroth, O. Schütze, M. Wiecek, Y. Jin, and C. Grimme. Lecture Notes in Computer Science 10173. Springer, Cham, pp. 77–87.

[18] Bökler, F., M. Ehrgott, C. Morris, and P. Mutzel (2017). "Output-Sensitive Complexity for Multiobjective Combinatorial Optimization." In: *Journal of Multi-Criteria Decision Analysis* 24(1–2), pp. 25–36.

[19] Bökler, F. and P. Mutzel (2015). "Output-Sensitive Algorithms for Enumerating the Extreme Nondominated Points of Multiobjective Combinatorial Optimization Problems." In: *Algorithms – ESA 2015*. Ed. by N. Bansal and I. Finocchi. Lecture Notes in Computer Science 9294. Springer, Berlin, Heidelberg, pp. 288–299.

[20] Bökler, F. and P. Mutzel (2017). "Tree-Deletion Pruning in Label-Correcting Algorithms for the Multiobjective Shortest Path Problem." In: *WALCOM: Algorithms and Computation. WALCOM 2017*. Ed. by S. H. Poon, M. Rahman, and H. C. Yen. Lecture Notes in Computer Science 10167. Springer, Cham, pp. 190–203.

[21] Borndörfer, R., S. Schenker, M. Skutella, and T. Strunk (2016). "PolySCIP." In: *Mathematical Software – ICMS 2016*. Ed. by G. M. Greuel, T. Koch, P. Paule, and A. Sommese. Lecture Notes in Computer Science 9725. Springer, Cham, pp. 259–264.

[22] Bremner, D. (1996). "Incremental convex hull algorithms are not output sensitive." In: *Algorithms and Computation. ISAAC 1996*. Ed. by T. Asano, Y. Igarashi, H. Nagamochi, S. Miyano, and S. Suri. Lecture Notes in Computer Science 1178. Springer, Berlin, Heidelberg, pp. 26–35.

[23] Bremner, D. D. (1997). "On the computation of vertex and facet enumeration for convex polytopes." PhD thesis. McGill University, Canada.

[24] Bremner, D., K. Fukuda, and A. Marzetta (1998). "Primal—Dual Methods for Vertex and Facet Enumeration." In: *Discrete and Computational Geometry* 20(3), pp. 333–357.

[25] Brumbaugh-Smith, J. and D. Shier (1989). "An empirical investigation of some bicriterion shortest path algorithms." In: *European Journal of Operational Research* 43(2), pp. 216–224.

[26] Brunsch, T. and H. Röglin (2015). "Improved Smoothed Analysis of Multiobjective Optimization." In: *Journal of the ACM* 62(1), 4:1–4:58.

[27] Burton, B. A. and M. Özlen (June 2010). "Projective geometry and the outer approximation algorithm for multiobjective linear programming." arXiv:1006.3085 [math.OC].

[28] Cerqueus, A., A. Przybylski, and X. Gandibleux (2015). "Surrogate upper bound sets for bi-objective bi-dimensional binary knapsack problems." In: *European Journal of Operational Research* 244(2), pp. 417–433.

[29] Chankong, V. and Y. Haimes (1983). *Multiobjective Decision Making: Theory and Methodology*. Elsevier Science, New York.

[30] Chazelle, B. (1993). "An optimal convex hull algorithm in any fixed dimension." In: *Discrete and Computational Geometry* 10(4), pp. 377–409.

[31] Cherkassky, B. V., A. V. Goldberg, and T. Radzik (1996). "Shortest paths algorithms: Theory and experimental evaluation." In: *Mathematical Programming* 73(2), pp. 129–174.

[32] Chimani, M. and K. Klein (2010). "Algorithm Engineering: Concepts and Practice." In: *Experimental Methods for th Analysis of Optimization Algorithms*. Ed. by T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss. Springer, Berlin, Heidelberg. Chap. 6, pp. 131–158.

[33] Cobham, A. (1965). "The intrinsic computational difficulty of functions." In: *Proceedings of the 1964 International Congress on Logic, Methodology and Philosophy of Science*. Ed. by B.-H. Yehoshua. Studies in logic and the foundations of mathematics. North-Holland Publishing Company, Amsterdam, pp. 24–30.

[34] Cohon, J. L. (1978). *Multiobjective Programming and Planning*. Academic Press, New York.

[35] Cook, S. A. (1971). "The complexity of theorem-proving procedures." In: *Theory of Computing. STOC 1971*. ACM, New York, pp. 151–158.

[36] Cormen, T. H., C. E. Leiserson, R. L. Rivest, and C. Stein (2009). *Introduction to Algorithms*. MIT Press, Cambridge, Massachusetts, USA.

[37] Creignou, N., A. Meier, J.-S. Müller, J. Schmidt, and H. Vollmer (2013). "Paradigms for Parameterized Enumeration." In: *Mathematical Foundations in Computer Science*. Lecture Notes in Computer Science 8087. Springer, Berlin, Heidelberg, pp. 290–301.

[38] Csirmaz, L. (2016). "Using multiobjective optimization to map the entropy region." In: *Computational Optimization and Applications* 63(1), pp. 45–67.

[39] Dantzig, G. B. (1990). "Origins of the simplex method." In: *A history of scientific computing*. Ed. by S. G. Nash. ACM, New York, pp. 141–151.

[40] Dauer, J. (1993). "On degeneracy and collapsing in the construction of the set of objective values in a multiple objective linear program." In: *Annals of Operations Research* 46(2), pp. 279–292.

[41] Dauer, J. and R. Gallagher (1996). "A combined constraint-space, objective-space approach for determining high-dimensional maximal efficient faces of multiple objective linear programs." In: *European Journal of Operational Research* 88(2), pp. 368–381.

[42] Dauer, J. and Y. Liu (1990). "Solving multiple objective linear programs in objective space." In: *European Journal of Operational Research* 46(3), pp. 350–357.

[43] Dauer, J. and O. Saleh (1990). "Constructing the set of efficient objective values in multiple objective linear programs." In: *European Journal of Operational Research* 46(3), pp. 358–365.

[44] Delling, D. and D. Wagner (2009). "Pareto Paths with SHARC." In: *Experimental Algorithms. SEA 2009*. Ed. by J. Vahrenhold. Lecture Notes in Computer Science 5526. Springer, Berlin, Heidelberg, pp. 125–136.

[45] Diakonikolas, I. (2011). "Approximation of Multiobjective Optimization Problems." PhD thesis. Columbia University.

[46] Diakonikolas, I. and M. Yannakakis (2008). "Succinct approximate convex Pareto-curves." In: *ACM Symposium on Discrete Algorithms. SODA 2008*. Ed. by S.-H. Teng. ACM, New York, pp. 74–83.

[47] Diakonikolas, I. and M. Yannakakis (2009). "Small Approximate Pareto Sets for Biobjective Shortest Paths and Other Problems." In: *SIAM Journal on Computing* 39(4), pp. 1340–1371.

[48] Dial, R. B. (1979). "A model and algorithm for multicriteria route-mode choice." In: *Transportation Research Part B* 13(4), pp. 311–316.

[49] Dumitrescu, I. and N. Boland (2001). "Algorithms for the weight constrained shortest path problem." In: *International Transactions in Operational Research* 8, pp. 15–29.

[50] Dyer, M. E. (1983). "The Complexity of Vertex Enumeration Methods." In: *Mathematics of Operations Research* 8(3), pp. 381–402.

[51] Ecker, J., N. Hegner, and I. Kouada (1980). "Generating all maximal efficient faces for multiple objective linear programs." In: *Journal of Optimization Theory and Applications* 30(3), pp. 353–381.

[52] Ecker, J. and I. Kouada (1978). "Finding all efficient extreme points for multiple objective linear programs." In: *Mathematical Programming* 14(1), pp. 249–261.

[53] Edmonds, J. (1965). "Paths, trees and flowers." In: *Canadian Journal of Mathematics* 17, pp. 449–467.

[54] Ehrgott, M. (2000). "Approximation algorithms for combinatorial multicriteria optimization problems." In: *International Transactions in Operational Research* 7(1), pp. 5–31.

[55] Ehrgott, M. (2005). *Multicriteria Optimization*. Springer, Berlin, Heidelberg.

[56] Ehrgott, M. (2006). "A discussion of scalarization techniques for multiple objective integer programming." In: *Annals of Operations Research* 147(1), pp. 343–360.

[57] Ehrgott, M. and X. Gandibleux (2000). "A survey and annotated bibliography of multiobjective combinatorial optimization." In: *OR Spektrum* 22(4), pp. 425–460.

[58] Ehrgott, M. and X. Gandibleux (2007). "Bound sets for biobjective combinatorial optimization problems." In: *Computers & Operations Research* 34(9), pp. 2674–2694.

[59] Ehrgott, M., A. Löhne, and L. Shao (2012). “A dual variant of Benson’s "outer approximation algorithm" for multiple objective linear programming.” In: *Journal of Global Optimization* 52(4), pp. 757–778.

[60] Emelichev, V. A. and V. A. Perepelitsa (1992). “On cardinality of the set of alternatives in discrete many-criterion problems.” In: *Discrete Mathematics and Application* 2(5), pp. 461–471.

[61] Erb, S., M. Kobitzsch, and P. Sanders (2014). “Parallel Bi-objective Shortest Paths Using Weight-Balanced B-trees with Bulk Updates.” In: *Experimental Algorithms. SEA 2014*. Ed. by J. Gudmundsson and J. Katajain. Lecture Notes in Computer Science 8504. Springer, Cham.

[62] Evans, J. and R. Steuer (1973). “A revised simplex method for linear multiple objective programs.” In: *Mathematical Programming* 5(1), pp. 54–72.

[63] Figueira, J. R., C. Fonseca, P. Halffmann, K. Klamroth, L. Paquete, S. Ruzika, B. Schulze, M. Stigelmayr, and D. Willems (2017). “Easy to say they’re hard, but hard to say they are easy – Towards a categorization of tractable multiobjective combinatorial optimization problems.” In: *Journal of Multi-Criteria Decision Analysis* 24, pp. 82–98.

[64] Fredman, M. L. and R. E. Tarjan (1987). “Fibonacci heaps and their uses in improved network optimization algorithms.” In: *Journal of the ACM* 34(3), pp. 596–615.

[65] Fukuda, K. and A. Prodon (1996). “Double description method revisited.” In: *Combinatorics and Computer Science*. Ed. by M. Deza, R. Euler, and I. Manoussakis. Lecture Notes in Computer Science 1120. Springer, Berlin, Heidelberg, pp. 91–111.

[66] Gal, T. (1977). “A general method for determining the set of all efficient solutions to a linear vectormaximum problem.” In: *European Journal of Operational Research* 1(5), pp. 307–322.

[67] Galand, L., A. Ismaili, P. Perny, and O. Spanjaard (2013). “Bidirectional Preference-based Search for Multiobjective State Space Graph Problems.” In: *Combinatorial Search. SoCS 2013*. Ed. by M. Helmert and G. Röger. AAAI Press, Palo Alto, USA, pp. 80–88.

[68] Ganley, J. L., M. J. Golin, and J. S. Salowe (1995). “The multi-weighted spanning tree problem.” In: *Computing and Combinatorics. COCOON 1995*. Ed. by D. Du and M. Li. Lecture Notes in Computer Science 959. Springer, Berlin, Heidelberg, pp. 141–150.

[69] Garey, M. R. and D. S. Johnson (1979). *Computers and Intractability; A Guide to the Theory of NP-Completeness*. Ed. by V. Klee. A Series of Books in the Mathematical Sciences. W. H. Freeman & Co., New York.

[70] Grötschel, M., L. Lovász, and A. Schrijver (1991). *Geometric Algorithms and Combinatorial Optimization*. Algorithms and Combinatorics 2. Springer.

[71] Guerriero, F. and R. Musmanno (Dec. 2001). “Label Correcting Methods to Solve Multicriteria Shortest Path Problems.” In: *Journal of Optimization Theory and Applications* 111(3), pp. 589–613.

[72] Hamacher, H. W. and G. Ruhe (1994). "On spanning tree problems with multiple objectives." In: *Annals of Operations Research* 52(4), pp. 209–230.

[73] Hamel, A. H., A. Löhne, and B. Rudloff (2014). "Benson type algorithms for linear vector optimization and applications." In: *Journal of Global Optimization* 59(4), pp. 811–836.

[74] Hansen, P. (1979). "Bicriterion Path Problems." In: *Multiple Criteria Decision Making Theory and Application*. Ed. by G. Fandel and T. Gal. Lecture Notes in Economics and Mathematical Systems 177. Springer, Berlin, Heidelberg, pp. 109–127.

[75] Heyde, F. and A. Löhne (2008). "Geometric Duality in Multiple Objective Linear Programming." In: *SIAM Journal of Optimization* 19(2), pp. 836–845.

[76] Hoaglin, D. C., F. Mosteller, and J. W. Turkey (1983). *Understanding Robust and Exploratory Data Analysis*. John Wiley & Sons, Hoboken, USA.

[77] Horoba, C. (2010). "Exploring the Runtime of an Evolutionary Algorithm for the Multi-Objective Shortest Path Problem." In: *Evolutionary Computation* 18(3), pp. 357–381.

[78] Isermann, H. (1977). "The Enumeration of the Set of All Efficient Solutions for a Linear Multiple Objective Program." In: *Operational Research Quarterly* 28(3), pp. 711–725.

[79] Johnson, D. S., M. Yannakakis, and C. H. Papadimitriou (1988). "On generating all maximal independent sets." In: *Information Processing Letters* 27(3), pp. 119–123.

[80] Kallay, M. (1986). "Convex hull made easy." In: *Information Processing Letters* 22(3), p. 161.

[81] Karmarkar, N. (1984). "A new polynomial-time algorithm for linear programming." In: *Combinatorica* 4(4), pp. 373–395.

[82] Karp, R. M. (1972). "Reducibility Among Combinatorial Problems." In: *Complexity of Computer Computations*. Ed. by R. E. Millers, J. W. Thatcher, and J. D. Bohlinger. The IBM Research Symposia Series. Springer, Boston, Massachusetts, USA, pp. 85–103.

[83] Kellerer, H., U. Pferschy, and D. Pisinger (2004). *Knapsack Problems*. Springer, Berlin, Heidelberg.

[84] Khachiyan, L. G. (1979). "A polynomial algorithm in linear programming (in Russian)." In: *Doklady Akademii Nauk SSSR* 244, pp. 1093–1096.

[85] Khachiyan, L., E. Boros, K. Borys, K. Elbassioni, and V. Gurvich (2008). "Generating All Vertices of a Polyhedron Is Hard." In: *Discrete and Computational Geometry* 39(1–3), pp. 174–190.

[86] Kirkpatrick, D. G. and R. Seidel (1985). "Output-size sensitive algorithms for finding maximal vectors." In: *Computational Geometry. SCG 1985*. Ed. by J. O'Rourke. ACM, New York, pp. 89–96.

[87] Klamroth, K., R. Lacour, and D. Vanderpooten (2015). "On the representation of the search region in multi-objective optimization." In: *European Journal of Operational Research* 245(3), pp. 767–778.

[88] Knuth, D. (1976). “Big Omicron and big Omega and big Theta.” In: *ACM SIGACT News* 8(2), pp. 18–24.

[89] Korte, B. and J. Vygen (2012). *Combinatorial Optimization: Theory and Algorithms*. Algorithms and Combinatorics 21. Springer, Berlin, Heidelberg.

[90] Kung, H. T., F. Luccio, and F. P. Preparata (Oct. 1975). “On Finding the Maxima of a Set of Vectors.” In: *Journal of the ACM* 22(4), pp. 469–476.

[91] Ladner, R. (1975). “On the Structure of Polynomial Time Reducibility.” In: *Journal of the ACM* 22(1), pp. 155–171.

[92] Laumanns, M., L. Thiele, and E. Zitzler (2006). “An efficient, adaptive parameter variation scheme for metaheuristics based on the epsilon-constraint method.” In: *European Journal of Operational Research* 169(3), pp. 932–942.

[93] Lawler, E. L., J. K. Lenstra, and A. H. G. Rinnooy Kan (Aug. 1980). “Generating All Maximal Independent Sets: NP-Hardness and Polynomial-Time Algorithms.” In: *SIAM Journal on Computing* 9(3), pp. 558–565.

[94] Levin, L. (1973). “Universal Search Problems (in Russian).” In: *Problems of Information Transmission* 9(3), pp. 115–116.

[95] Leys, C., C. Ley, O. Klein, P. Bernard, and L. Licata (2013). “Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median.” In: *Journal of Experimental Social Psychology* 49(4), pp. 764–766.

[96] Löhne, A. and B. Weißing (2016). “Equivalence between polyhedral projection, multiple objective linear programming and vector linear programming.” In: *Mathematical Methods of Operations Research* 84(2), pp. 411–426.

[97] Löhne, A. and B. Weißing (2017). “The vector linear program solver Bensolve – notes on theoretical background.” In: *European Journal of Operational Research* 260(3), pp. 807–813.

[98] Martins, E. Q. V. (1984). “On a Multicriteria Shortest Path Problem.” In: *European Journal of Operational Research* 16(2), pp. 236–245.

[99] Mohamed, C., J. Bassem, and L. Taicir (2010). “A genetic algorithm to solve the bicriteria shortest path problem.” In: *Electronic Notes in Discrete Mathematics* 36, pp. 851–858.

[100] Müller-Hannemann, M., F. Schulz, D. Wagner, and C. Zaroliagis (2007). “Timetable Information: Models and Algorithms.” In: *Algorithmic Methods for Railway Optimization*. Ed. by F. Geraets, L. Kroon, A. Schoebel, D. Wagner, and C. D. Zaroliagis. Lecture Notes in Computer Science 4359. Springer, Berlin, Heidelberg, pp. 67–90.

[101] Mullesgaard, K., J. L. Pedersen, H. Lu, and Y. Zhou (2014). “Efficient Skyline Computation in MapReduce.” In: *International Conference on Extending Database Technology (EDBT)*.

[102] Nemhauser, G. L. and Z. Ullmann (1969). “Discrete Dynamic Programming and Capital Allocation.” In: *Management Science* 15(9), pp. 494–505.

[103] Nikolova, E., J. A. Kelner, M. Brand, and M. Mitzenmacher (2006). “Stochastic Shortest Paths via Quasi-convex Maximization.” In: *Algorithms – ESA 2006. ESA 2006*. Ed. by Y. Azar and T. Erlebach. Lecture Notes in Computer Science 4168. Springer, Berlin, Heidelberg, pp. 552–563.

[104] Okamoto, Y. and T. Uno (2007). "A Polynomial-Time-Delay and Polynomial-Space Algorithm for Enumeration Problems in Multi-criteria Optimization." In: *Algorithms and Computation. ISAAC 2007*. Ed. by T. Tokuyama. Lecture Notes in Computer Science 4835. Springer, Berlin, Heidelberg, pp. 609–620.

[105] Özpeynirci, Ö. (2008). "Approaches for Multiobjective Combinatorial Optimization Problems." PhD thesis. Middle East Technical University.

[106] Özpeynirci, Ö. and M. Köksalan (2010). "An Exact Algorithm for Finding Extreme Supported Nondominated Points of Multiobjective Mixed Integer Programs." In: *Management Science* 56(12), pp. 2302–2315.

[107] Paixao, J. M. and J. L. Santos (2007). *Labeling Methods for the General Case of the Multi-Objective Shortest Path Problem - A Computational Study*. Tech. rep. 07-42. Universidade de Coimbra.

[108] Paixao, J. and J. Santos (2009). "Labeling Methods for the General Case of the Multiobjective Shortest Path Problem: A Computational Study." In: *Computational Intelligence and Decision Making*. Ed. by A. Madureira, C. Reis, and V. Marques. Intelligent Systems, Control and Automation: Science and Engineering 61. Springer, Dordrecht, pp. 489–502.

[109] Papadimitriou, C. H. (2012). "The New Faces of Combinatorial Optimization." In: *Combinatorial Optimization. ISCO 2012*. Ed. by A. R. Mahjoub, V. Markakis, I. Milis, and V. T. Paschos. Lecture Notes in Computer Science 7422. Springer, Berlin, Heidelberg, pp. 19–23.

[110] Papadimitriou, C. H. and K. Steiglitz (1998). *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, New York, USA.

[111] Papadimitriou, C. H. and M. Yannakakis (2000). "On the Approximability of Trade-Offs and Optimal Access of Web Sources." In: *Symposium on Foundations of Computer Science. FoCS 2000*. Ed. by A. Blum. IEEE, Washington, USA, pp. 86–92.

[112] Philip, J. (1972). "Algorithms for the vector maximization problem." In: *Mathematical Programming* 2(1), pp. 207–229.

[113] Philip, J. (1977). "Vector maximization at a degenerate vertex." In: *Mathematical Programming* 13(1), pp. 357–359.

[114] Przybylski, A., X. Gandibleux, and M. Ehrgott (2010). "A Recursive Algorithm for Finding All Nondominated Extreme Points in the Outcome Set of a Multiobjective Integer Programme." In: *INFORMS Journal on Computing* 22(3), pp. 371–386.

[115] Raith, A. and M. Ehrgott (2009). "A Comparison of Solution Strategies for Biobjective Shortest Path Problems." In: *Computers & OR* 36(4), pp. 1299–1331.

[116] Rockafellar, R. T. and R. J.-B. Wets (2009). *Variational Analysis*. Springer, Berlin, Heidelberg.

[117] Rothvoss, T. (2014). "The matching polytope has exponential extension complexity." In: *ACM Symposium on Theory of Computing. SToC 2014*. Ed. by D. Shmoys. ACM, New York, pp. 263–272.

[118] Rudloff, B., F. Ulus, and R. Vanderbei (2017). “A Parametric Simplex Algorithm for Linear Vector Optimization Problems.” In: *Mathematical Programming* 163(1–2), pp. 213–242.

[119] Ruhe, G. (1988). “Complexity results for multicriterial and parametric network flows using a pathological graph of Zadeh.” In: *Zeitschrift für Operations Research* 32(1), pp. 9–27.

[120] Sanders, P. and L. Mandow (2013). “Parallel Label-Setting Multi-Objective Shortest Path Search.” In: *Parallel & Distributed Processing. IPDPS 2013*. Ed. by S. Ranka. IEEE, Washington DC, USA, pp. 215–224.

[121] Schmidt, J. (2009). “Enumeration: Algorithms and Complexity.” MA thesis. Leibniz Universität Hannover.

[122] Schönfeld, K. (1964). “Effizienz und Dualität in der Aktivitätsanlyse (German).” PhD thesis. Freie Universität Berlin.

[123] Seidel, R. (1995). “The upper bound theorem for polytopes: an easy proof of its asymptotic version.” In: *Computational Geometry* 5(2), pp. 115–116.

[124] Serafini, P. (1986). “Some Considerations about Computational Complexity for Multi-Objective Combinatorial Problems.” In: *Recent Advances and Historical Development of Vector Optimization*. Ed. by J. Jahn and W. Krabs. Lecture notes in economics and mathematical systems 294. Springer, Berlin, pp. 222–231.

[125] Shekelyan, M., G. Jossé, and M. Schubert (2015). “ParetoPrep: Efficient Lower Bounds for Path Skylines and Fast Path Computation.” In: *Advances in Spatial and Temporal Databases. SSTD 2015*. Ed. by C. Claramunt, M. Schneider, R. C.-W. Wong, L. Xiong, W.-K. Loh, C. Shahabi, and K.-J. Li. Lecture Notes in Computer Science 9239. Springer, Cham.

[126] Shekelyan, M., G. Jossé, M. Schubert, and H.-P. Kriegel (2014). “Linear Path Skyline Computation in Bicriteria Networks.” In: *Database Systems for Advanced Applications. DASFAA 2014*. Ed. by S. S. Bhowmick, C. E. Dyreson, C. S. Jensen, M. L. Lee, A. Muliantara, and B. Thalheim. Lecture Notes in Computer Science 8421. Springer, Cham, pp. 173–187.

[127] Shephard, G. C. (1968). “A theorem on cyclic polytopes.” In: *Israel Journal of Mathematics* 6(4), pp. 368–372.

[128] Skriver, A. J. V. (2000). “A Classification of Bicriterion Shortest Path Algorithms.” In: *Asia-Pacific Journal of Operational Research* 17(2), pp. 199–212.

[129] Spielman, D. A. and S.-H. Teng (2009). “Smoothed Analysis: An Attemplt to Explain the Behavior of Algorithms in Practice.” In: *Communications of the ACM* 52(10), pp. 76–84.

[130] Steuer, R. E. (1975). “ADBASE: A Program for Analyzing Multiple Objective Linear Programming Problems.” In: *Journal of Marketing Research* 12, pp. 453–455.

[131] Strijbosch, L., A. V. Doorne, and W. Selen (1991). “A simplified MOLP algorithm: The MOLP-S procedure.” In: *Computers & Operations Research* 18(8), pp. 709–716.

[132] Tarapata, Z. (2007). "Selected Multicriteria Shortest Path Problems: An Analysis of Complexity, Models and Adaptation of Standard Algorithms." In: *International Journal of Applied Mathematics and Computer Science* 17(2), pp. 269–287.

[133] Tiwary, H. R. (2008). "Complexity of Some Polyhedral Enumeration Problems." PhD thesis. Universität des Saarlandes.

[134] Tsaggouris, G. and C. D. Zaroliagis (2009). "Multiobjective Optimization: Improved FPTAS for Shortest Paths and Non-Linear Objectives with Applications." In: *Theory of Computing Systems* 45(1), pp. 162–186.

[135] Tung, C. T. and K. L. Chew (1992). "A multicriteria Pareto-optimal path algorithm." In: *European Journal of Operational Research* 62(2), pp. 203–209.

[136] Ulungu, E. L. and J. Teghem (1994). "The two-phases method: An efficient procedure to solve bi-objective combinatorial optimization problems." In: *Foundations of Computing and Decision Sciences* 20(2), pp. 149–165.

[137] Vassilvitskii, S. and M. Yannakakis (2005). "Efficiently computing succinct trade-off curves." In: *Theoretical Computer Science* 348(2–3), pp. 334–356.

[138] Warburton, A. (1987). "Approximation of Pareto Optima in Multiple-Objective Shortest-Path Problems." In: *Operations Research* 35(1), pp. 70–79.

[139] Yu, P. and M. Zeleny (1975). "The set of all nondominated solutions in linear cases and a multicriteria simplex method." In: *Journal of Mathematical Analysis and Applications* 49(2), pp. 430–468.

[140] Yu, P. and M. Zeleny (1976). "Linear multiparametric programming by multicriteria simplex method." In: *Management Science* 23(2), pp. 159–170.

[141] Zeleny, M. (1974). *Linear Multiobjective Programming*. Lecture Notes in Economics and Mathematical Systems 95. Springer, Berlin.

[142] Zitzler, E., L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fonseca (2002). *Performance Assessment of Multiobjective Optimizers: An Analysis and Review*. Tech. rep. 139. Institut für Technische Informatik und Kommunikationsnetze, ETH Zürich.

Part III.

Appendix

Index

$[n]$, 7
$\leq$, 7
$\preceq$, 7
$\otimes_c$, 8
*, 10
$\mathcal{D}$, 56
$\boldsymbol{e}_i$, 8
$\operatorname{int} M$, 9
$\mathcal{K}_d$-maximal, 56
$\lambda(\boldsymbol{\ell})$, 56
ℓ_p-norm, 8
lexmin, 8
min, 7
$\max_{\mathcal{K}_d} M$, 56
max, 7
$N_P(\boldsymbol{x})$, 9
$\mathcal{P}$, 18
$\mathcal{P}_{s,t}^G$, 10
$\operatorname{rec} P$, 9
$\operatorname{ri} M$, 9
S_n, 7
$\mathcal{T}_G$, 10
$T_P(\boldsymbol{x})$, 9
$\operatorname{vert} P$, 9
W_d^0, 48
$W_{\mathcal{P}}(\boldsymbol{y})$, 48
$W_{\mathcal{P}}^0(\boldsymbol{y})$, 48
$\operatorname{wmin} M$, 8

Alphabet, 10
Arc, 9

Biobjective Unconstrained Optimization Problem, 17
Boundary, 9
BUCO, 17

Canonical Decision Problem
  Multiobjective, 27
  Single Objective, 13
co-NP, 14
Combinatorial Optimization Problem, 13
Compact Formulation, 62
Computational Problems, 11
Cone, 9
  Extreme Ray, 9
Convex Relaxation, 39
Cycle, 10
  Hamiltonian, 10

Decision Problem, 11
  Complement, 11
Delay, 33
DelayP, 34

ε-approximation algorithm, 103
ε-Constraint Scalarization, 36
ε-cover, 103
ε-Dominance, 103
ε-Pareto set, 103
Edge, 9
Efficient Solution, 15
Enumeration Algorithm, 33
Enumeration Problem, 33
  Configurations, 33
  Instance, 33

Finished Decision Problem, 70
FPTAS, 104

Graph, 9
  bipartite, 10
  complete, 10
  digraph, 9

directed, 9

Half-space, 9

Ideal Point, 16
Image, 7
IncP, 33
Incremental Delay, 34
Incremental Polynomial Time, 33
Integer Hull, 9
Interior, 9

Language, 10
Linear Combination, 8
Lower Image, 56

Matching, 17
Mathematical Optimization Problem, 12
    Instance, 12
    Objective Function, 12
    Solution, 12
Maximum Norm, 8
MO-Flow, 19
MO-Matching, 17
MO-Z-Flow, 19
MOAP, 17
MOCO, 16
MOLP, 18
MOLP Relaxation, 39
MOMC, 16
MOO, 15
MOSP, 17
MOST, 17
MOTSP, 17
MUCO, 16
Multiobjective (global) Minimum Cut Problem, 16
Multiobjective Assignment Problem, 17
Multiobjective Combinatorial Optimization Problem, 16
Multiobjective Integer Minimum Cost Flow Problem, 19
Multiobjective Linear Programming Problem, 18
Multiobjective Matching Problem, 17
Multiobjective Optimization Problem, 15
    Instance, 15
    Solution, 15
Multiobjective Rational Minimum Cost Flow Problem, 19
Multiobjective Shortest Path Problem, 17
Multiobjective Spanning Tree Problem, 17
Multiobjective Travelling Salesperson Problem, 17
Multiobjective Unconstrained Optimization Problem, 16

No-Instance, 11
Node, 9
Nondominated, 8
Nondominated Set, 15
    Extreme Points, 16
Normal Cone, 9
NP, 13

Output-sensitive, 34
Output-Sensitive Reduction, 77

P, 13
Pareto-front, 15
Pareto-optimal Solution, 15
Partition, 7
    k-Partition, 7
Path, 10
    Length, 10
    Simple, 10
Permutation, 7
Polar Dual, 9
Polyhedron, 9
    Dimension, 9
    Extreme Points, 9
    Faces, 9
    Facets, 9
    Recession Cone, 9
    Recession Direction, 9
Polynomial Delay, 34
Polynomial Time Delay, 34
Polynomial Time Delay with Polynomial Space, 34
Polynomial-Time Equivalence, 15

PSDelayP, 34

RAM, 11
Random Access Machine, 11
Reduction, 14
    Cook, 14
    Karp, 14
    Output-sensitive, 77
Relative Interior, 9

Spanning Tree, 10
Star-operator, 10
String, 10
Subgraph, 10
    Spanning, 10
    Tree, 10

Tangent Cone, 9
TotalP, 33
Tree, 10
Tree Deletion Pruning, 96

Unit Vector, 8
Upper Image, 18

Value Vector, 16

Weakly nondominated, 8
Weakly Pareto-optimal Solution, 15

Yes-Instance, 11

List of Figures

6.1. Showing the reduction in the proof for Theorem 6.2. Arcs with no label have cost **0**. 71

9.1. Comparison of the running times (in seconds) of the label-selection (LS) and node-selection (NS) strategies on the instances by Paixao and Santos [108]. 94
9.2. Comparison of the running times (in seconds) of the label-selection (LS) and node-selection (NS) strategies on real-world road networks by PTV AG. 95
9.3. Illustration of the tree-deletion pruning 96
9.4. Measuring how many nodes have been touched which could have been deleted by tree-deletion pruning in the label-selection (LS) and node-selection (NS) strategies on the instances by Paixao and Santos [108] . 100
9.5. Comparison of the running times (in seconds) of the node-selection strategy with (NS-TD) and without (NS) tree-deletion pruning on the instances by Paixao and Santos [108] 101
9.6. Comparison of the running times (in seconds) of the node-selection strategy with (NS-TD) and without (NS) tree-deletion pruning on real road networks by PTV AG 102

10.1. An example for the definition of $\mathbf{1}+\varepsilon$-cover 104
10.2. The cell structure introduced by Papadimitriou and Yannakakis [111]. 106
10.3. Running times of the approximation algorithms on the real world road networks by PTV AG. 116
10.4. Running times and empirical ratios of the approximation algorithms on the instances by Paixao and Santos [108] 117
10.5. Running times and ratios of the approximation algorithms on the `CompleteN-large` instances by Paixao and Santos [108] 118
10.6. Running times and ratios of the approximation algorithms on the `GridN-large` instances by Paixao and Santos [108] 119
10.7. Running times and ratios of the approximation algorithms on the `RandomN-large` instances by Paixao and Santos [108] 120
10.8. Running times and ratios of the approximation algorithms on the instances by Paixao and Santos [108] based on different ε 122
10.9. Running times and ratios of the approximation algorithms on the instances by Paixao and Santos [108] based on different ε 123

List of Tables

3.1. State of well-known MOP problems 24

5.1. Computational results on multiobjective assignment instances with d objectives and n resources. Instances with an * were taken from the work by Özpeynirci and Köksalan [106]. 67
5.2. Comparison of the dual Benson implementation with and without lexicographic oracle on multiobjective assignment instances with $d = 5$. 67

9.1. Overview on the artificial test instances by Paixao and Santos [108] . . 90
9.2. Running times and nondominated set sizes of the node-selection (NS) implementation and the node-selection implementation with tree-deletion (NS-TD) on the power grid optimization instances. In the penultimate instance, the NS implementation exceeded the memory limit of 16 GB. 98

10.1. Running times and empirical ratios of the approximation algorithms on the power transmission grid optimization instances. 115

List of Listings

1. Benson's Outer Approximation Algorithm 45
2. Dual Variant of Benson's Outer Approximation Algorithm 57

3. Abstract version of the LS algorithm 92
4. Abstract version of the NS algorithm 92

5. A Generic MOSP Labeling Algorithm 107