BERGISCHE
UNIVERSITÄT
WUPPERTAL

Multi-Objective Combinatorial Optimization: Supportedness, Representations, and Integer Network Flows

Dissertation
zur Erlangung des Doktorgrades (Dr. rer. nat.)
Fakultät für Mathematik und Naturwissenschaften
Bergische Universität Wuppertal

vorgelegt von
David Könen

Erstgutachter: PD Dr. Michael Stiglmayr
Zweitgutachter: Prof. Dr. Luís Paquete
Drittgutachter: Prof. Dr. Stefan Ruzika

Acknowledgements

This acknowledgment is a tribute to all those who have supported me in writing this thesis or been part of my life as a doctoral student.

First and foremost, I would like to thank my supervisor, Michael Stiglmayr, for supporting me with numerous productive discussions and ideas. I always felt that your office door was open and that you were there to support me even when you had a lot to do yourself, or in times when I came in every ten minutes.

I am also highly grateful to Kathrin Klamroth, for welcoming me into her Optimization Group at the University of Wuppertal. For me, it felt like the perfect place to work on my PhD. Thank you for your encouragement, patience, and valuable feedback over the last few years. I always felt that you supported me and believed in my abilities, even when I made mistakes.

During my time at the University of Wuppertal, I had the freedom to work on my own ideas, yet I always felt supported whenever I faced challenges or needed guidance. I am truly thankful for having you both as advisors.

I am grateful to Luís Paquete for his dedication as a reviewer of this thesis. During my time as PhD student, I had the opportunity to visit Coimbra multiple times. Each visit was met with a warm welcome from Luís and the entire team at the Department of Informatics Engineering at the University of Coimbra. These experiences were always productive and invaluable. I would also like to extend my thanks to Stefan Ruzika for his dedication and effort as a reviewer of this thesis.

I thank Andreas Frommer and Matthias Bolten for being on my doctorate committee.

Furthermore, I thankfully acknowledge partial financial support of the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), project number 441310140.

In addition, I want to thank Daniela Gaul and Arne Heimendahl for proofreading.

I would also like to thank Christiane Spisla and Daniel Schmidt for collaborating on my first research project, which laid the foundation for my thesis. Both of them supported me in my decision to start a PhD, and I am very grateful to them for the positive start they provided me.

Moreover, I extend my thanks to all my colleagues, Lara Löhken, Julia Sudhoff-Santos, Daniela Gaul, Britta Efkes, Claudia Totzeck, Julius Bauß, Muriel Berno, Onur Doganay, Fabian Heldmann, Renée Lamsfuß, Tobias Suszka, Mahmoud Khatab, Kirsten Wilshaus, Hanna Schilar, and the entire optimization group in Wuppertal for creating such an excellent working atmosphere and helping me in numerous questions about teaching or research. Thank you all for the beautiful and wildly entertaining time at conferences, such as in Creta, Coimbra, and Tunisia, and the entertaining conversations during coffee breaks or cold drinks after work.

I would also like to mention all my other fellow students, whose support has made my studies such an exciting and instructive phase of my life.

Moreover, I am also thankful to Michael Jünger, Rainer Schrader, and Jens Schmidt for their impactful lectures at the University of Cologne. The chance to work as a tutor for their lectures deepened my understanding and aroused my research interest in the first place.

Lastly, I would like to express my deepest gratitude to my family and all my friends. Having them in my life has immeasurable improved it. I could not imagine my life without them. This thesis would not have been possible without the support of the above-mentioned individuals.

Contents

1 Introduction

Combinatorial optimization focuses on selecting an optimal subset from a typically huge but finite set of discrete objects or options. These options often represent elements such as edges in a graph or a network and encompass several well-studied problem classes, including scheduling, facility location, graph partitioning, vehicle routing, and network flow problems. These problems have widespread applications in logistics, telecommunications, operations research, and various other fields.

Among these, this thesis primarily focuses on network flow problems, which arise when a commodity moves through an underlying network, creating a flow of resources or information. Nowadays, these networks are everywhere in our daily lives, appearing in various forms, from transportation, communication, and power distribution networks to supply chains and social interactions. Consider, for example, navigation systems finding the shortest or most sustainable path between your location and desired destination. The very influential textbook by Ahuja et al. [1993] presents over 150 applications of network flow problems across various fields such as engineering, management, and scientific domains. These problems are fundamental and well-studied in combinatorial optimization and have been addressed since the early 1950s [see, e.g. Ahuja et al., 1993; Bertsekas, 1998, and the references given therein].

The *minimum cost flow problem* involves moving a static flow through a network at minimal cost. The given network has specific nodes that supply the flow units while others demand it. The flow units are moved along the arcs of the network. The flow along these arcs is constrained by specified lower and upper bounds, known as capacities, and has associated costs per flow unit. The objective is to find a flow that satisfies supply, demand, and capacity constraints and minimizes the overall cost. The minimum cost flow problem has numerous real-world applications, especially in industry and decision-making, such as inventory planning, data scaling, lot sizing, location problems, DNA sequence alignment, and project management [Ahuja et al., 1993]. Moreover, the minimum cost flow problem contains several combinatorial optimization problems as special cases. Among others, the important *transportation problem* and the *assignment problem* reduce to the minimum cost flow problem. The problem can be described more mathematically in the following way. Given a directed graph with n nodes $V = \{v_1, \ldots, v_n\}$ and m arcs $A = \{e_1, \ldots, e_m\}$, non-negative arc costs and non-negative integer capacities, and an integer-valued supply/demand b_v for each node $v \in V$, the task is to determine a b-flow, or referred to as flow, with minimum total cost, i. e., a network flow that respects the capacities on all arcs $a \in A$ and has flow balance exactly b_v in each node $v \in V$. For the single-objective version of this problem, there are various polynomial algorithms. A comprehensive overview of the related literature for minimum cost flow problems is provided in Ahuja et al. [1993]. However, real-world problems often involve multiple conflicting objectives, and no solution optimizes all objectives simultaneously.

Imagine a group of friends from Germany planning a surf trip to the beautiful town of Ericeira, Portugal. They have several transportation options to reach Ericeira, including planes, cars, trains, buses, or combinations, each providing different costs, travel times, and environmental impacts. Planes and trains from Germany may only connect to larger nearby cities, such as Lisbon, requiring additional transportation afterward. Moreover, the group may not have enough car seats for everyone, or there may not be enough remaining tickets available for the cheapest flight or train, potentially forcing the group to split up and take separate routes. The group aims to find the cheapest, fastest, and most sustainable transportation. However, no possibility minimizes all three objectives. Arriving by plane in Lisbon and then renting cars might be the fastest option, but it is also the least sustainable. Driving with the own cars is likely the cheapest but slower option. On the other hand, only opting for trains and buses is the most sustainable option, but it comes with being slower and potentially more expensive. Therefore, the objectives conflict and a trade-off must be made between time, cost, and sustainability.

This scenario shows that these multi-objective problems appear in our daily lives. As stated in Ehrgott [2005]:

> *Life is about decisions. Decisions, no matter if made by a group or an individual, usually involve several conflicting objectives. The observation that real world problems have to be solved optimally according to criteria, which prohibit an "ideal" solution [...]* ***has led to the development of multicriteria optimization****.*

Multi-objective problems with conflicting objectives arise in many real-world problems, including logistics, economics, finance, and more. In those conflicting scenarios, no solution exists that optimizes all objectives simultaneously. In particular, one is interested in finding solutions with the property that it can only be improved with respect to one objective if at least one other objective deteriorates. Such a solution is called *efficient solution* or *Pareto-optimal* solution and its image is called *nondominated point*.

Various approaches exist for tackling multi-objective optimization problems. The primary approaches can be categorized as the following. In *a-priori methods*, the decision-maker specifies their preferences, such as weights or goals, before the optimization process begins. In *a-posteriori* methods, the optimization process generates a set of efficient solutions, allowing the decision-maker to select the most suitable one afterward. In *interactive methods*, the decision-maker interacts with the optimization process iteratively, refining preferences and guiding the search for the most suitable solution.

This thesis primarily focuses on a-posteriori methods, aiming to determine all or a suitable subset of the efficient solutions or nondominated points, for a summary of solution concepts in multi-objective optimization, see Serafini [1986]. One subset of interest could be the set of all *supported efficient solutions*, which are those efficient solutions that can be obtained as optimal solutions of a single-objective problem that optimizes a convex combination of the multiple objectives with weights strictly greater than zero. While multi-objective linear optimization problems only contain supported efficient solutions, the efficient solution set of multi-objective combinatorial optimization problems, such as

multi-objective network flow problems, also contain nonsupported efficient solutions. The nonsupported efficient solutions typically outnumber the supported ones, where the latter are more straightforward to determine, can serve as high-quality representations [Sayın, 2024], and can be used as a foundation for two-phase methods to generate the entire nondominated point set. Despite their importance, several characterizations for supported efficient solutions and thus supported nondominated points are used in the literature.

The *multi-objective integer minimum cost flow* problem is significantly more challenging than its single-objective counterpart. The multi-objective case has been reviewed in Hamacher et al. [2007b], where the authors comment on the lack of efficient algorithms. Multi-objective integer minimum cost flows, like any discrete multi-objective optimization problem, can be considered as *enumeration problem*. This involves enumerating all nondominated points or subsets of the nondominated point set with respect to other solution concepts. Determining the complete nondominated point set for multi-objective integer minimum cost flow problems is known to be intractable in the sense that the number of nondominated points grows exponentially with the problem size [Hansen, 1980]. Furthermore, the corresponding *canonical decision problem*, i.e., finding a feasible flow that has cost lower or equal to a given value for each objective, is **NP**-complete.

Optimization problems are often classified based on whether they can be solved in polynomial or exponential time. However, many multi-objective discrete optimization problems are intractable, meaning that the cardinality of the nondominated point set grows exponentially with respect to the input size [Hansen, 1980; Hamacher and Ruhe, 1994; Ehrgott, 2005, ...]. Even when finding a single solution may be easy, or the corresponding canonical decision problem may be solvable in polynomial time, enumerating all desired solutions of such an intractable problem in polynomial time is impossible. Therefore, another approach is needed to classify the "hardness" of such intractable enumeration problems. For such problems, one may seek algorithms with a running time that can be bounded by a polynomial in both the input and the output size. Such algorithms are known as *output-polynomial time* algorithms. For a detailed discussion of output-sensitive complexity, we refer to the survey Johnson et al. [1988]. Such an algorithm, or proof that no such algorithm can exist (unless $\mathbf{P} = \mathbf{NP}$), can indicate whether the intractable enumeration problem can be considered as "easy" or "hard".

One goal of this thesis is to determine whether output-polynomial time algorithms exist for different solution concepts for the multi-objective integer minimum cost flow problem and to verify if the supported nondominated points yield high-quality representations of the complete nondominated point set.

Outline and Contribution of this thesis

This thesis presents new algorithms to determine alternative solutions and various subsets of the efficient set for the multi-objective integer minimum cost flow problem. It analyzes the output sensitivity complexity with respect to different solution concepts. The thesis is a step toward addressing the open question of whether output-polynomial time algorithms exist for these different solution concepts. The thesis shows how to compute the complete nondominated point set as well as specific subsets, such as the *supported* nondominated

points, and evaluates the quality of the supported nondominated points as representations of the entire nondominated point set. Furthermore, while several characterizations exist for supported efficient solutions, the thesis summarizes equivalent definitions and characterizations for supported efficient solutions. It introduces a distinction between supported and weakly supported efficient solutions.

This thesis is structured as follows: Chapter 2 sets up the notation and terminology and provides essential definitions and foundational concepts for the subsequent chapters. This includes an introduction to computational complexity, polyhedral theory, linear programming, multi-objective optimization, graph theory, and the minimum cost flow problem.

Chapter 3 presents scalarization-based algorithms for multi-objective combinatorial optimization problems. It focuses on methods that decompose the overall problem into a series of scalarized single-objective subproblems, which can then be solved using available single-objective (IP-)solvers. The presented methods are generic, i.e., they are independent of specific problem structures and hence generally applicable. Furthermore, the chapter references foundational results and algorithms, establishing connections with various aspects such as search region, search zones, local upper bounds, defining points, complexity results, and redundancy avoidance. It emphasizes the importance of these generic approaches and underscores their efficiency in solving multi-objective combinatorial optimization problems while exploring a wide range of options for their implementation.

Chapter 4 addresses an inconsistency in various definitions of supported nondominated points within multi-objective combinatorial problems (MOCO). Despite the importance of supportedness, several different characterizations for supported efficient solutions (and supported nondominated points) are used in the literature. While these definitions are equivalent for multi-objective linear problems, they can yield different sets of supported nondominated points for MOCO problems. The chapter shows by an example that these definitions are not equivalent for MOCO or general multi-objective optimization problems. Moreover, the structural and computational properties of the resulting sets of supported nondominated points are analyzed. Equivalent definitions and characterizations for supported efficient solutions are summarized, and a distinction between *supported* and *weakly supported* efficient solutions is introduced.

While multi-objective linear optimization problems only contain supported nondominated points, the nondominated set of multi-objective combinatorial optimization problems, such as integer network flow problems, may also contain weakly supported and nonsupported nondominated points. These points generally outnumber the supported ones and are more challenging to determine, as they cannot be obtained as optimal solutions of weighted sum problems with weights strictly greater than zero. Chapter 5 considers the supported nondominated points as representations for the complete nondominated point set in network flow problems. Various quality metrics, such as coverage error, hypervolume ratio, and ε-indicator, are presented and used to analyze and compare the quality of these representations. Multiple classes of network flow problems are generated to evaluate the representations. The results indicate that the supported nondominated points consistently provide high-quality representations.

Chapter 6 addresses the problem of determining all optimal integer solutions of a linear integer network flow problem, referred to as the *all optimal integer flow (AOF)* problem. A new algorithm is derived with improved time complexity compared to existing state-of-the-art algorithms for determining the set of all optimal integer flows. Our improvement is made possible by replacing the shortest path sub-problem with a more efficient way to determine a so-called proper zero cost cycle using a modified depth-first search technique. This algorithm serves as a foundational tool for the solution methods presented in Chapter 7 and Chapter 8.

As a byproduct, the analysis yields an enhanced algorithm to determine the k best integer flows. Furthermore, lower and upper bounds for the number of all optimal integer and feasible integer solutions are established based on the fact that any optimal solution can be derived from an initial optimal *tree solution* combined with a conical combination of incidence vectors of all *induced cycles* with bounded coefficients.

Chapter 7 addresses the problem of enumerating all *supported efficient solutions* for multi-objective integer minimum cost flow problems, providing insights into the time complexity for enumerating all supported nondominated points and all supported efficient solutions. An output-polynomial time algorithm to determine all supported efficient solutions is derived. Moreover, it proves that the existence of an output-polynomial time algorithm to determine all weakly supported nondominated points (or all weakly supported efficient solutions) for a multi-objective integer minimum cost flow problem with a fixed number of $d \geq 3$ objectives can be excluded unless $\mathbf{P} = \mathbf{NP}$. It also shows that there cannot exist an output-polynomial time algorithm for the enumeration of all supported nondominated points that determine the points in a lexicographically ordered way in the outcome space unless $\mathbf{P} = \mathbf{NP}$.

Chapter 8 proposes novel methods for identifying supported nondominated points in bi-objective minimum cost flow problems accompanied by a numerical comparison between decision- and objective-space methods. A novel, equivalent, and more compact formulation of the minimum cost flow ILP formulation used in the ε-constraint scalarization approach is introduced, demonstrating enhanced efficiency in the numerical tests.

Chapter 9 concludes the thesis by summarizing the main contributions and remarks for future research.

Credits and Publications

Some parts of this thesis are the result of joint work which has been published, submitted, or will be submitted:

- Chapter 3 is based on a book chapter, co-authored with Kathrin Klamroth and Kerstin Dächert, which appears in Dächert et al. [forthcoming].
- Chapter 4 is the result of joint work with Michael Stiglmayr and is available as a technical report in Könen and Stiglmayr [2025b], which has been accepted for publication in the Journal of Multi-Criteria Decision Analysis.

- The content of Chapter 5 is based on joint work with Lara Löhken, Michael Stiglmayr and Kathrin Klamroth and will be submitted.
- The content of Chapter 6 is based on the joint work with Daniel Schmidt and Christiane Spisla and has been published in Könen et al. [2022a].
- The results presented in Chapter 7 and Chapter 8 are the outcome of joint work with Michael Stiglmayr. They have resulted in a publication Könen and Stiglmayr [2025a] and a technical report Könen and Stiglmayr [2023], which has been submitted to the Journal of Combinatorial Optimization.
- Chapter 2 sets the preliminaries for the following chapter and contains parts from all articles and the book chapter.

2 Preliminaries

This chapter states all the necessary basic definitions and notations used throughout the thesis. Its purpose is to set a consistent and coherent notation.

Key conventions include:

- Scalars and vectors are represented by lowercase Latin or Greek letters (e.g., y, and λ, ε). Vectors are distinguished by superscripts (e.g., x^1, x^2) and components of vectors by subscripts (e.g., x_1, x_2).
- Sets are denoted by Roman or Greek capital letters (e.g., A, B, and Λ, Σ), exceptions are the general notation of the basic number sets $\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}$, displaying the sets of natural, integer, rational and real numbers, respectively. The non-negative orthant of $\mathbb{R}^p$ is denoted by $\mathbb{R}^p_{\geqq} := \{x \in \mathbb{R}^p : x \geqq 0\}$ and analogously its interior $\mathbb{R}^p_{>}$ and $\mathbb{R}^p_{\geq} := \{x \in \mathbb{R}^p : x \geqq 0, x \neq 0\}$. Sets displaying a feasible decision or outcome space of an optimization problem are displayed by calligraphical letters (e.g., $\mathcal{X}, \mathcal{Y}$).
- Matrices and problems are also indicated with Roman capital letters (e.g., A, E). Derived problems are distinguished by superscript variants (e.g., E^{FIN}).

The reader is assumed to be familiar with basic concepts of linear and integer programming, network optimization, graph theory, and combinatorial optimization. However, the key concepts used in the thesis are stated below. All definitions and results here are standard and can be found, along with their proofs, in various texts books on combinatorial optimization, notably those by Ahuja et al. [1993], Nemhauser and Wolsey [1999], Schrijver [2003], and Korte and Vygen [2012]. If a theorem is presented without a proof, the proof can be found in the corresponding reference. When a proof is provided for a theorem with cited reference, it is based on the proof in the source (although it may differ slightly; compare proofs as needed).

This chapter first introduces computational problems, algorithms, and their complexity in Section 2.1. Section 2.2 introduces linear and integer linear optimization and polyhedral theory. Next, in Section 2.3, the scope is extended to multi-objective optimization problems Section 2.4, covers essential definitions from graph theory, and introduces the most well-known graph problems relevant to the thesis. Finally, Section 2.5 addresses the central problem class of network flow problems and their related issues.

2.1 Computational Complexity

In this section, computational problems, the concepts of input and output, and operations involved in algorithms are formalized. Also, the necessary definitions for classifying the complexity of algorithms are provided. This section is primarily based on the textbooks by Papadimitriou [1994], Schrijver [2003], and Korte and Vygen [2012].

2.1.1 Algorithm Design and Computational Problems

Informally, a computation problem defines a desired input-output relationship, and an algorithm is a well-defined procedure that details how to achieve this relationship by transforming the input values into output values through a sequence of operations.

Alphabet, Language The *input* of an algorithm refers to the provided data for processing, while the *output* is the result produced by the algorithm. Both can take various forms, such as numbers, characters, or abstract objects like graph nodes, and can be represented or stored as finite strings of *bits* (0's and 1's). For example, a sorting algorithm takes a list of numbers as input and outputs the list sorted in non-decreasing order.

An *alphabet* Σ is a finite set of such *symbols* or *characters*. This thesis considers only the *binary alphabet* given by the set $\{0, 1\}$, where a *string* or *word* over Σ is a sequence of elements from the alphabet. The set Σ^* consists of all possible strings over the alphabet Σ. A *language* is a subset $I \subseteq \Sigma^*$ over the alphabet Σ.

The *encoding length* of a data structure is the length of the string s required to represent it, denoted by $|s|$. The encoding length of a natural number $n \in \mathbb{N}$ represented by its binary representation is bounded by $\lceil \log(n+1) \rceil$. For an integer $z \in \mathbb{Z}$, the encoding length is $\lceil \log_2(|z|+1) \rceil + 1$ to account for the sign. A rational number $r \in \mathbb{Q}$, expressed as $r = p/q$ with $p, q \in \mathbb{Z}$, $q \geq 1$, and greatest common divisor $\gcd(p, q) = 1$, has an encoding length of $1 + \lceil \log(|p|+1) \rceil + \lceil \log q \rceil$. For a vector $x \in \mathbb{Q}^n$ or a matrix $A \in \mathbb{Q}^{m \times n}$, the encoding length is the sum of the encoding lengths of its components. The encoding of complex structures as graphs will be discussed in the respective Section 2.4.

Random Access Machine The complexity of an algorithm or computational problem is defined with respect to a computational (or machine) model like a *turing machine* model or *random access machine* (RAM) model. This thesis adopts the RAM model, which consists of a central processing unit (CPU) (single processor) with a finite set of instructions and a memory array where each cell can be accessed directly in constant time (random-access memory). The instructions are executed sequentially, without concurrent operations, including arithmetic operations, data movement, and control instructions, each of which takes a constant amount of time. This computational model is an abstraction that allows algorithms to be compared based on performance by counting the number of basic operations performed. RAMs are equivalent to Turing machines in polynomial time complexity. An algorithm that runs on a RAM in polynomial time implies the existence of a polynomial-time constrained Turing machine, see Papadimitriou [1994]. We refer to Cormen et al. [2001] and Sipser [2012] for an concise definition and overwiev of RAMs.

Polynomial Time Solvability Since the RAM is the fixed algorithmic model in this thesis, any reference to an algorithm implies a RAM-based algorithm. An algorithm is referred to as *polynomial time* algorithm if the number of steps is bounded by a polynomial in the input size $|x|$ for $x \in I \subseteq \Sigma^*$. Here, a step or *operation* consists of performing a single instruction. This concept will be formalized further to analyze the *running time* of algorithms in more detail.

Order of Growth The *order of growth* of a function describes its asymptotic behavior, providing characterizations of the upper bound on the running time of algorithms.

Definition 2.1 (Korte and Vygen, 2012)**.** *Let $f, g\colon I \to \mathbb{R}_>$ be two functions. The function f is in $\mathcal{O}(g)$, if there exist constants $\alpha, \beta > 0$ such that $f(x) \leq \alpha g(x) + \beta$ for all $x \in I$.*

For two words $s, x \in \Sigma^*$, the length $|s|$ of s is said to be in poly($|x|$) if there exists a polynomial p such that $|s| \in \mathcal{O}(p(|x|))$.

Running-time The worst-case time complexity of an algorithm provides an upper bound for the running time of any input, and it is used throughout this thesis to analyze the complexity of algorithms.

Definition 2.2 (Korte and Vygen, 2012)**.** *Let A be an algorithm that accepts inputs from a set of instances (language) $I \subseteq \Sigma^*$, and let $f : \mathbb{N} \to \mathbb{R}_>$. If there exists a constant α such that A terminates after at most $\alpha f(|x|) + \beta$ elementary steps for any $x \in I$, then A is said to run in time $\mathcal{O}(f)$, also refereed to as the* running-time *or* time complexity *of A.*

Let $n = |x|$ be the size of the input, $\bar{n}$ the number of elements of the input, $\tilde{n}$ the largest numeric value in the input, and $k \in \mathbb{Z}_>$. Then A is said to run in:

- *linear time* if A has time complexity $\mathcal{O}(n)$,
- *polynomial time* if A has time complexity $\mathcal{O}(n^k)$,
- *strongly polynomial time* if A has time complexity $\mathcal{O}(\bar{n}^k)$, and
- *pseudo polynomial time* if A has time complexity $\mathcal{O}(\tilde{n}^k)$.

If an algorithm A computes the output $f(x) \in Y$ for each input $x \in I \subseteq \Sigma^*$, then A is said to *compute* the function $f\colon X \to Y$. A function is *computable in polynomial time* if some polynomial time algorithm computes it.

Decision Problem A *decision problem* is a problem that can be posed as a yes/no question regarding the input values. This type of problem is central to studying algorithms and complexity because it simplifies the analysis by reducing problems to their core components. It will be shown that optimization problems can be reduced to decision problems, and we can define the complexity classes with regard to decision problems.

Definition 2.3. *A* decision problem *is a pair $D = (I, Y)$ such that*

1. *$I \subseteq \Sigma^*$ is the set of instances for some fixed alphabet Σ, and*
2. *$Y \subseteq I$ is the set of* yes-instances *and $I \backslash Y$ is the set of* no-instances*.*

Then, for a given word (input) $x \in I$, the problem is to determine whether $x \in Y$.

An example of a decision problem is the *subset-sum problem*, defined as follows:

Definition 2.4. *Given a set $N = \{1, \dots, n\}$ of n items with positive integer weights $w_1, \dots, w_n$ and a real value k, the* subset sum problem (SSP) *is to find a subset of N such that the corresponding total weight is exactly equal k. The formal definition is given by*

$$\sum_{j=1}^{n} w_j \, x_j = k$$
$$x_j \in \{0,1\} \quad \forall j \in \{1, \dots, n\}$$

also written as

$$\Big\{(w,k) \colon w^\top x = k, x \in \{0,1\}^n\Big\}. \tag{SSP}$$

Definition 2.5. *A decision algorithm for a decision problem D is a random access machine that computes the function $f \colon I \to \{0,1\}$ such that*

1. *on input $x \in I$ it outputs $f(x) = 1$ for $x \in Y$ and $f(x) = 0$ for $x \in I \backslash Y$, and*
2. *on every input terminates after a finite number of steps.*

If such a decision algorithm exists, a decision problem D is called decidable. *If the algorithm runs in polynomial time, D is called* polynomial decidable.

The set of instances $I \subseteq \Sigma^*$ is said to be (polynomial) decidable if the corresponding decision problem $D^* = (\Sigma^*, I)$ is (polynomial) decidable. For a decision problem $D = (I, Y)$, it is always assumed that I is polynomial decidable.

Optimization Problem Optimization problems, particularly combinatorial and multi-objective optimization problems, are the primary object of this thesis. This section introduces the classical concept of single-objective optimization. Since maximization can be defined analogously to minimization, all terms will be introduced for minimization, with the understanding that the corresponding definitions for maximization are analogous.

Definition 2.6. *An optimization problem consists of a set of instances and is defined for an instance by:*

$$\min_{x \in \mathcal{X}} \; f(x), \tag{OP}$$

where $f \colon \mathbb{R}^n \to \mathbb{R}$ is a real valued objective function *and $\mathcal{X} \subseteq \mathbb{R}^n$ is the* set of feasible solutions *to a given instance.*

For each optimization problem, a *corresponding decision problem* can be defined as: Given an optimization problem $\min_{x \in \mathcal{X}} f(x)$ and a constant $k \in \mathbb{Z}$, then the *corresponding decision problem* asks if there is a feasible solution $x \in X$ with value $f(x) \leq k$. An example of an optimization problem is the *knapsack problem*, defined as follows:

Definition 2.7. *Given a set $N = \{1, \dots, n\}$ of n items, each with a positive integer weight $w_1, \dots, w_n$ and a positive integer profit $p_1, \dots, p_n$, and a real value capacity k_1, the*

knapsack problem *(KP) is to find a subset of N such that the total weight does not exceed k_1 and the total profit is maximized. The formal definition is given by*

$$\max_{x \in \mathcal{X}} \sum_{j=1}^{n} p_j x_j$$

where $\mathcal{X} = \{x \in \{0,1\}^n \colon \sum_{j=1}^{n} w_j x_j \leq k_1\}$.

The corresponding decision problem is then defined as

$$\left\{ \left(w, p, k_1, k_2\right) \colon w^\top x \leq k_1, p^\top x \geq k_2, x \in \{0,1\}^n \right\}. \tag{KP}$$

These definitions do not depend on the input encoding. However, since this thesis considers the complexity of algorithms, the above definitions can also be stated more algorithmically, as in Korte and Vygen [2012] or Bökler [2018].

Definition 2.8 (Bökler, 2018)**.** *An (algorithmic) optimization problem is a triple* $O = (I, S, v)$ *where*

1. *a set of instances* $I \subseteq \Sigma^*$,
2. *a mapping* $S \colon I \to 2^{\Sigma^*}$ *which maps an instance* $x \in I$ *to its set of feasible solutions* S_x *with, and*
3. *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 $x \in I$*, it must hold that for each* $s \in S_x$*, the length* $|s|$ *is in poly*$(|x|)$*. It is assumed 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* $v(x, s') \leq v(x, s)$.

Note that this definition reduces the mapping from a real-valued function to a rational-valued function since computational representation only can represent rational numbers.

Definition 2.9 (Bökler, 2018)**.** *The* corresponding decision problem *to an optimization problem is*

$$\{(x, k) \in I \times \mathbb{Q} \colon \exists s \in S_x \text{ with } v(x, s) \leq k\},$$

or more formal the tuple $(I \times \mathbb{Q}, Y)$ *with* $(x, k) \in Y$ *where* $(x, k) \in I \times \mathbb{Q}$ *if there exists* $s \in S_x$ *with* $v(x, s) \leq k$.

The existence of a polynomial time algorithm for the decision problem implies that the corresponding optimization problem can also be solved in polynomial time by employing techniques like binary search. This linkage allows many optimization problems to be tackled efficiently when their corresponding decision problems are known to be solvable in polynomial time, see Sipser [2012].

2.1.2 Complexity Classes

For many combinatorial optimization problems, polynomial-time algorithms are known from literature. However, there are also problems for which no such algorithms are known. Although we cannot prove the non-existence of a polynomial-time algorithm, it is possible to classify problems as "hard" based on computational complexity theory. Complexity classes are used to categorize decision problems rather than the corresponding algorithms according to their computational complexity.

P and NP One of the most important class is the set **P**, which is refereed to as the set of *efficiently* solvable problems. Another important complexity class is **NP**.

Definition 2.10. *The set* **P** *is the set of all decision problems for which a polynomial time algorithm exists.*

Definition 2.11 (Schrijver, 2003)**.** *The set* **NP** *is the set of all decision problems* $D = (I, Y)$ *for which there exists a decision problem* $D' = (I', Y')$ *such that* $x \in Y$ *if and only if there exists word* w *with size bounded by poly*$(|x|)$ *such that* $(x, w) \in Y'$*. The word* w *is called a certificate for* x*, and an algorithm* A *for* D' *is referred to as* certificate-checking algorithm*.*

NP stands for nondeterministically polynomial time, since the string w could be chosen by the algorithm by guessing. So, guessing leads to a polynomial time algorithm. Informally, the class **NP** can be described as the set of all decision problems that can be solved by a polynomial time nondeterministic algorithm. It holds $\mathbf{P} \subseteq \mathbf{NP}$, since an algorithm for D' outputs x by deleting w and then applies the algorithm for D as a certificate-checking algorithm. However, whether $\mathbf{P} \neq \mathbf{NP}$ is unknown, making this one of the most important open problems in complexity theory. However, it is strongly believed that they are not equal.

Reducibility and Hardness Reducibility is used to show relationships between problems by transforming one problem into another and showing that the problem is not easier than the other.

Definition 2.12. *A decision problem* $D' = (I', Y')$ *can be* polynomial reduced *to a decision problem* $D = (I, Y)$ *if there exists a function* $f\colon I' \to I$ *computable in polynomial time such that* $f(x) \in Y$ *if* $x \in Y'$ *and* $f(x) \in I \backslash Y$ *if* $x \in I' \backslash Y'$*. We write* $D' \leq_P D$*.*

Proposition 2.13 (Korte and Vygen, 2012)**.** *If* D' *polynomially reduces to* D *and if there is a polynomial time algorithm for* D*, then there is a polynomial time algorithm for* D'*.*

Definition 2.14. *A decision problem* D *is called* **NP**-hard *if all other problems in* **NP** *polynomially transform to* D*. If* $D \in \mathbf{NP}$ *and* D *is* **NP***-hard the problem is called* **NP**-complete*.*

Stephen Cook introduces the concept of **NP**-completeness in his famous work Cook [1971], where he proved that the *satisfiability problem* is **NP**-complete, by demonstrating that the satisfiability problem is both in **NP** and **NP**-hard. In Karp [1972], other **NP**-complete

problems are presented, by proving that they can be polynomially reduced to the satisfiability problem, thereby establishing their equivalent computational complexity.

Theorem 2.15 (Karp, 1972)**.** *The subset sum problem* (SSP) *is* **NP***-complete.*

Theorem 2.16 (Karp, 1972)**.** *The knapsack problem* (KP) *is* **NP***-complete.*

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

Definition 2.17. *For a decision problem $D = (I, Y)$, its* complement *is defined as the decision problem $(I, I \backslash Y)$. The class* **co-NP** *consists of all problems whose complements are in* **NP***. A decision problem $\mathcal{P} \in$* **co-NP** *is called* **co-NP**-complete *if all other problems in* **co-NP** *polynomially transform to $\mathcal{P}$.*

Theorem 2.18 (Korte and Vygen, 2012)**.** *A decision problem is* **co-NP***-complete if and only if its complement is* **NP***-complete.*

2.1.3 Output-polynomial Complexity

This section formally introduces the theory of *output-sensitive* complexity of enumeration problems. For a comprehensive introduction, see Johnson et al. [1988]. The concepts discussed here are based on Bökler [2018].

Enumeration Problem While decision problems seek the existence of a solution to some problem instances, in many parts of this thesis, the interest lies in finding multiple solutions, including scenarios where alternative optimal solutions for optimization problems are sought or in multi-objective optimization where subsets of efficient solutions are requested. Such problems can be defined as *enumeration problems*, where we aim to output all solutions.

Definition 2.19. *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 $\operatorname{poly}(|x|)$,*

where Σ^* can be interpreted as the set of all finite strings over $\{0, 1\}$.

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

Definition 2.20. *An* enumeration algorithm *for an enumeration problem $E = (I, C)$ is a* random access machine *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.*

Definition 2.21. *An enumeration algorithm for an enumeration problem $E = (I, C)$ is said to run in* output-polynomial time (is output-sensitive) *if its running time is in* $\text{poly}(|x|, |C(x)|)$ *for $x \in I$.*

Definition 2.22. *An enumeration problem is called* intractable *if the cardinality of the configuration set is exponential in the size of the instance.*

Finished Decision Problem A finished decision problem E^{FIN} for an enumeration problem $E = (I, C)$ is defined as follows: Given an instance $x \in I$ of the enumeration problem and a subset $M \subseteq C(x)$ of the configuration set, the goal is to decide if $M = C(x)$, i.e., to determine if all configurations have been found.

Theorem 2.23 (Lawler et al., 1980)**.** *If the enumeration problem E can be solved in output-polynomial time, then $E^{\text{FIN}} \in \mathbf{P}$.*

2.2 Polyhedral Theory and Linear Programming

This section summarizes the necessary definitions of polyhedral theory and linear programming, mainly based on the textbooks by Nemhauser and Wolsey [1999] and Schrijver [2003].

2.2.1 Polyhedral Theory

This thesis uses the notation $A+B \coloneqq \{a+b\colon a \in A, b \in B\}$ and $A \cdot B \coloneqq \{a \cdot b\colon a \in A, b \in B\}$ for the Minkowski sum and the Minkowski product of two sets $A, B \subseteq \mathbb{R}^d$, respectively. For a set C, we define the boundary of C by ∂C. In the following, let n and m be non-negative scalars.

Hyperplane, Half-space The subset $\{x \in \mathbb{R}^n\colon Ax = b\} \subseteq \mathbb{R}^n$, with $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ is called an *affine subspace* of $\mathbb{R}^n$. A *hyperplane* in $\mathbb{R}^n$ is a subset

$$H = \{x \in \mathbb{R}^n\colon a^\top x = b\} \subseteq \mathbb{R}^n$$

for $a \in \mathbb{R}^n \backslash \{0\}$ and $b \in \mathbb{R}$, representing an affine subspace of dimension $n-1$. The vector a is called the corresponding *normal vector* to the hyperplane. The set $\{x \in \mathbb{R}^n\colon a^\top x \leq b\}$ defines a *half-space*, whose boundary is the corresponding hyperplane H.

Polyhedron and Polytope A *polyhedron* is the intersection of a finite number of half-spaces. Formally, it can be written as the set

$$P = \{x \in \mathbb{R}^n\colon Ax \leqq b\}$$

with $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. Sets of the form $\{x \in \mathbb{R}^n\colon Ax \geqq b\}$ are also polyhedra, since they can be rewritten as $\{x \in \mathbb{R}^n\colon (-A)x \leqq -b\}$. Intersecting finitely many polyhedra results in a polyhedron; therefore, the set $\{x \in \mathbb{R}^n\colon Ax = b\}$ is also a polyhedron since it represents the intersection of $\{x \in \mathbb{R}^n\colon Ax \geqq b\}$ and $\{x \in \mathbb{R}^n\colon Ax \leqq b\}$. Half-spaces, the

empty set, and the entire space $\mathbb{R}^n$ also qualify as polyhedra, with $\emptyset = \{x \in \mathbb{R}^n \colon 0^\top x = 1\}$ and $\mathbb{R}^n = \{x \in \mathbb{R}^n \colon 0^\top x = 0\}$. A polyhedron P is called *rational* if there is a matrix $A \in \mathbb{Q}^{m \times n}$ and a vector $b \in \mathbb{Q}^m$ such that $P = \{x \in \mathbb{R}^n : Ax \leqq b\}$. In the following, if not otherwise mentioned, only rational polyhedra are considered.

A *polytope* is a bounded polyhedron. Formally, for a polytope P, there exits $r > 0$ such that $P \subseteq \{x \in \mathbb{R}^n \colon \|x\| < r\}$. A polytope can be equivalently described as the *convex hull* of a finite set of points $X = \{x^1, \dots x^k\}$, i.e.,

$$P = \operatorname{conv}\{x^1, x^2, \dots, x^k\} := \left\{ \sum_{i=1}^{k} \lambda_i x^i \colon x^i \in X, \lambda \in \mathbb{R}^k, k \in \mathbb{N}, \lambda_i \geq 0, \sum_{i=1}^{k} \lambda_i = 1 \right\},$$

see e.g., Nemhauser and Wolsey [1999].

Face An inequality $a^\top x \leq b$ is *valid* for a polyhedron P if it is satisfied by all $x \in P$, or equivalently, if $P \subseteq \{x \in \mathbb{R}^n \colon a^\top x \leq b\}$. Let $a^\top x \leq b$ a valid inequality of P, then $F = \{x \in P : a^\top x = b\}$ is called a *face* of P. Note that both P and $\emptyset$ are faces of P. If $F := \{x \in P : a^\top x = b\} \neq \emptyset$, then $\{x \in \mathbb{R}^n : a^\top x = b\}$ is a *supporting hyperplane*. In this context, $a^\top x = b$ *induces* or *supports* the face F. Each face F is itself a polyhedron. A non-empty face F of P is a *facet* of P if $\dim F = \dim P - 1$. A face with $\dim F = 1$ is called an *edge*. The face F is a *vertex* or *supported solution* of P if $\dim F = 0$.

Interior Point, Ray An *interior point* of a polyhedron $P \subseteq \mathbb{R}^n$ is a point $x \in P$ that lies strictly inside the polyhedron, meaning there exists a small positive radius $\varepsilon > 0$ such that the open ball $B(x, \varepsilon) = \{y \in \mathbb{R}^n \colon \|y - x\| < \varepsilon\}$ is entirely contained within P. Formally, x is an interior point of P if $B(x, \varepsilon) \subseteq P$ for some $\varepsilon > 0$. If $x \in P$ is not an interior point, it is said to lie on the boundary of P. In the context of polyhedra, a *ray* $r \in \mathbb{R}^n \setminus \{0\}$ is a direction in which the polyhedron extends infinitely, i.e., for a polyhedron $P \subseteq \mathbb{R}^n$, a ray r is a direction such that $x + \lambda r \in P$ for any $\lambda \geqq 0$ and $x \in P$. If a ray is a face of the polyhedron, it is called *extreme ray*.

2.2.2 Linear Programming

A *linear program* (LP) problem involves optimizing a linear objective function over a polyhedron.

Definition 2.24. *A* linear program *problem is defined as*

$$\min_{x \in \mathcal{X}} c^\top x \tag{LP}$$

with the feasible region $\mathcal{X} = \{x \in \mathbb{R}^n \colon Ax \leqq b, x \geqq 0\}$ *containing the set of all* feasible solutions *and* $x \in \mathcal{X}$ *representing a vector of* decision variables. *The vector* $c \in \mathbb{R}^n$ *is the vector of* cost coefficients *for the linear* objective function $c^\top x$, *and* $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$ *describes the* m *linear constraints defining the feasible region* $\mathcal{X}$. *The space* $\mathbb{R}^n$ *is referred to as the decision space and* $\mathbb{R}$ *is the objective space.*

Including different types of constraints, such as equality constraints, into a polyhedral representation is straightforward. The extension of the non-negativity constraint $x_i \geq 0$ for all $i = 1, \dots n$ can be done without loss of generality since two variables $x_i^+ - x_i^- = x_i$ with $x_i^+, x_i^- \in \mathbb{R}_{\geqq}$ can model an unbounded variable $x_i \in \mathbb{R}$. To maximize an objective function $c^\top x$ it is equivalent to minimize $-c^\top x$ instead.

Optimal Solution A LP is *infeasible* if the polyhedron $\mathcal{X}$ of the feasible region is empty, meaning no points $x \in \mathbb{R}^n_{\geqq}$ satisfy all the constraints simultaneously. A LP problem is called *unbounded* if the objective function can be arbitrarily decreased within the feasible region, i.e., if for all $M \in \mathbb{R}$ there $\exists x \in \mathcal{X}$ such that $c^\top x < M$. In this case, the optimization direction $-c$ is a ray of the polyhedron.

Theorem 2.25 (Nemhauser and Wolsey, 1999)**.** *If a LP is neither infeasible nor unbounded, then it attains the optimum at a vertex of the polyhedron $\mathcal{X}$.*

Throughout this thesis, we are often interested in determining more than one optimal solution if alternative optimal solutions exist. The following statement will help identify all optimal solutions for a LP.

Theorem 2.26. *The set of all optimal solutions of a LP that is neither unbounded nor infeasible is attained in a face $F = H \cap \mathcal{X}$ of the polytope $\mathcal{X}$ and a supporting hyperplane H of $\mathcal{X}$.*

Proof. Since $\mathcal{X}$ is neither infeasible or unbounded, there exists $y^* = \min_{x \in \mathcal{X}} \{c^\top x\}$. It holds that $c^\top x \geqq y^*$ for all $x \in \mathcal{X}$, and there exists at least one x^* with $c^\top x^* = y^*$. Therefore, $F = \{x \in \mathcal{X} \colon c^\top x = y^*\}$ is a non-empty face of $\mathcal{X}$. Thus, all optimal solutions are given by F. □

Duality The dual problem to a linear program can serve as a certificate for optimality and is a fundamental concept in optimization theory. It establishes a relationship between a given optimization problem and a related problem, known as the dual problem. The study of duality is particularly useful in deriving bounds on the optimal solution, providing optimality conditions, and developing efficient algorithms.

Definition 2.27. *For a LP* $\min_{x \in \mathbb{R}^n} \{c^\top x \colon Ax \leqq b, x \geqq 0\}$ *the* dual program *(DP) problem is given by*

$$\max_{y \in \mathbb{R}^m} \{b^\top y \colon A^\top y \leqq c, y \leqq 0\}.$$

The dual problem (DP) is also a linear programming problem and is referred to as the *dual* of the *primal* LP. A solution of DP is called *dual solution*. The dual of the DP is the primal LP. The dual program provides a lower bound on the objective function of the primal problem. Specifically, for any feasible solution y to the dual, the value $b^\top y$ is a lower bound on $c^\top x$ for any feasible solution x. This relationship is known as the *weak duality*.

Theorem 2.28 (Nemhauser and Wolsey, 1999)**.** *If $x \in \mathcal{X}$ is a feasible solution to the primal LP and y is a feasibly solution to the dual problem, then*

$$b^\top y \leq c^\top x.$$

Proof. Let x, y feasible solutions. Consider the primal constraint $Ax \leqq b$ if multiplied with $y \leqq 0$

$$y^\top Ax \geq y^\top b$$

is still valid for all $x \in \mathcal{X}$. Rewrite

$$b^\top y \leq x^\top A^\top y.$$

Since y is feasible it holds $A^\top y \leq c$ and $x \geq 0$ and hence

$$b^\top y \leq x^\top A^\top y \leq x^\top c = c^\top x.$$ □

These feasible solutions do not necessarily exist, in these cases, however, the weak duality theorem has immediate consequences.

Property 2.29. *For the primal LP and its DP, the following statements hold:*

1. *If the primal problem is unbounded, the dual problem is infeasible.*
2. *If the dual problem is unbounded, the primal problem is infeasible.*
3. *If the primal problem has a feasible solution x and the dual problem has a feasible solution y with $b^\top y = c^\top x$, then x and y are optimal solutions to the primal problem and dual problem, respectively.*

If both the primal and the dual program have a feasible solution, the bound provided by the dual program is tight, which is known as *strong duality*.

Theorem 2.30. *If the primal or dual problem has a finite optimal solution, so does the other and it holds*

$$\min_{x \in \mathbb{R}^n} \{c^\top x \colon Ax \leqq b, x \geqq 0\} = \max_{y \in \mathbb{R}^m} \{b^\top y : A^\top y \leqq c, y \leqq 0\}.$$

The following complementary slackness conditions, provide a direct relation of the two problems, a necessary and sufficient optimality condition for both the primal and dual problem.

Theorem 2.31 (Nemhauser and Wolsey, 1999)**.** *If x is a feasible solution of the primal LP $\min\{c^\top x \colon Ax \leqq b, x \geqq 0, x \in \mathbb{R}^n\}$ and y is a feasible solution for the dual given by $\max\{b^\top y \colon A^\top y \leqq c, y \leqq 0, y \in \mathbb{R}^m\}$ then x, y are optimal solutions if and only if*

- *for all $j = 1, \ldots, n$ with $x_j > 0$ it holds $\sum_{i=1}^m a_{ij} y_i = c_j$, and*
- *for all $i = 1, \ldots, m$ with $y_i < 0$ it holds $\sum_{j=1}^n a_{ij} x_j = b_i$.*

Note that all proofs can be found in the corresponding references.

Solution Methods Linear programming (LP) problems can be solved using a variety of solution methods, each with its own strengths and shortcomings, based on the specific characteristics and size of the LP problem at hand.

1. *Simplex Method:* Developed by Dantzig [1947], the simplex method is a procedure that iteratively moves from one vertex to another along the edges of the feasible region to find an optimal vertex. It is in practice and, on average, quite efficient, despite no polynomial time worst-case time complexity has been proved. The current pivot selection rules have been proven to have exponential worst-case time complexity. Moreover, there is also the interesting open question if the diameter of polyhedron can be exponential in its dimension and the number of facets. If so, even an "optimal" pivot rule would not ensure polynomial time complexity, as the length of the shortest path from the starting solution to the optimal solution might have an exponential number of facets.

2. *Interior Point Methods:* These methods, pioneered by Karmarkar [1984], approach the optimal solution from within the feasible region rather than traversing the boundary. They are polynomial-time algorithms and are highly effective for large-scale LP problems, competing with the simplex method. While the theoretically run-time is polynomial in contrast to the simplex method, which can be exponential in the worst case—in practice, they often perform comparably to the simplex method. In Renegar [1988], the first *path-following* method was introduced, where the search for the optimal solution follows a carefully designed trajectory through the interior of the feasible region, guided by a sequence of barrier functions. This approach has since become one of the most widely used variants of interior point methods, offering both strong theoretical guarantees and competitive practical performance.

3. *Primal-dual Methods:* First presented by Dantzig and Ford [1956] as another means of solving linear programs. These methods are iterative algorithms that simultaneously consider the primal and dual formulations. By iteratively improving both the primal and dual solutions while maintaining duality feasibility and primal near-feasibility to achieve the optimal solution. Starting with a dual feasible solution, the primal-dual method, given a current feasible dual solution, looks for a primal feasible solution that obeys the complementary slackness condition. While primal-dual methods have, in their original form, not survived as algorithms for linear programming, they have found widespread use for devising algorithms for some combinatorial optimization problems, such as network flow problems, where they exploit the network structure [Ford and Fulkerson, 1956; Ford and Fulkerson, 1962].

For a comprehensive overview of these solution methods and even more methods, such as the ellipsoid method, we refer to the textbooks by Schrijver [1998], Nemhauser and Wolsey [1999], and Korte and Vygen [2012].

Integer Linear Programming If in a LP it is further required that each component for the vector of decision variables is an integer, i.e., $x \in \mathbb{Z}^n$, we speak of an *integer linear program* (ILP).

Definition 2.32. *An* integer linear program *(ILP) problem is defined as*

$$\min_{x \in \mathcal{X}} c^\top x$$

with the feasible region $\mathcal{X} = \{x \in \mathbb{Z}^n \colon Ax \leqq b, x \geqq 0\} = P \cap \mathbb{Z}^n_{\geqq}$*. If only a subset of the variables is restricted to an integer value, the corresponding problem is called* mixed integer linear *(MILP) program. When the variables* $x \in \{0,1\}^n$ *model only binary decisions, the problem is called* binary integer linear program *(BILP).*

Unless **P** = **NP**, no polynomial-time algorithm exists that solves ILP in general [Karp, 1972]. However, the solution of the linear relaxation associated with an ILP, obtained by omitting the integrality condition on the variables x, can provide an approximation or lower bound for the optimal objective value of the ILP. Specific polynomial-time algorithms have been developed for particular classes of ILPs. These classes often relate to combinatorial problems having specific structural properties. Other solution concepts include:

1. *Branch and Bound Methods:* First presented by Land and Doig [1960], these methods gained significant attention in the literature and became widely adopted in discrete programming and optimization. These methods partition the feasible region recursively, generating smaller subproblems by bounding the optimal solution within each partition, e.g., by making use of the linear relaxations of an ILP. Subproblems can be discarded if their bound proves worse than the best-known feasible solution, effectively pruning the search space and improving efficiency. This strategy enables Branch and Bound to solve many ILPs to optimality, though its performance depends heavily on problem structure and the applied bounding techniques.

2. *Cutting Plane Methods:* These methods, first presented by Gomory [1958], iteratively tighten the relaxation of an ILP by systematically adding constraints (cutting planes) derived from violated inequalities found during the solution process. At each step, the method identifies a facet of the feasible region that is violated by the current solution, and a cutting plane is added to eliminate the infeasible region, progressively tightening the relaxation. This process continues until an optimal integer solution is found or further cuts are no longer effective.

We refer to the textbooks by Wolsey [1998] and Nemhauser and Wolsey [1999] for a comprehensive overview of these solution methods.

Integer Polyhedra and Total Unimodularity In specific cases, the structure of a polyhedron can exhibit integrality properties, in which the vertices of the polyhedron correspond to integer solutions. In such cases, the linear relaxation problem has an integer optimal solution.

Definition 2.33. *A polyhedron* P *is an* integer polyhedron *if each face of* P *contains at least one integer point, i.e., if and only if all vertices* v *of* P *are integer.*

Consider an ILP with a feasible region given by $\mathcal{X} = P \cap \mathbb{Z}^n$ where $P = \{x \in \mathbb{R}^n \colon Ax \leqq b\}$. If P is an integer polyhedron, it holds that

$$\min_{x \in \mathcal{X}} c^\top x = \min_{x \in P} c^\top x,$$

meaning the linear relaxation problem has an integer optimal solution, and we can obtain an optimal solution to the ILP by solving the linear relaxation problem. This characterizes integer polyhedra.

Theorem 2.34 (Schrijver, 1998)**.** *A rational polyhedron P in $\mathbb{Q}^n$ is integer if and only if for each $c \in \mathbb{Q}^n$ the value of $\max_{x \in P}\{c^\top x\}$ is an integer if it is finite.*

Total unimodularity of matrices is an important property for verifying if a polyhedron is integer.

Definition 2.35. *A matrix A is said to be* totally unimodular *if every square submatrix, defined by an arbitrary subset of row and column indices of A (of the same cardinality), has a determinant equal to $0, 1$, or -1. In particular, each entry of a totally unimodular matrix is $0, 1$, or -1.*

Theorem 2.36 (Schrijver, 1998)**.** *A polyhedron is an integer polyhedron if A is totally unimodular and $b \in \mathbb{Z}^m$.*

It follows that each LP that has integer data and a totally unimodular constraint matrix has integer optimum primal and dual solutions.

Property 2.37. *If $A \in \mathbb{R}^{m \times n}$ is totally unimodular and $b \in \mathbb{Z}^m$, $c \in \mathbb{Z}^n$, then the primal and dual problem*

$$\min_{x \in \mathbb{R}^n} \left\{ c^\top x : Ax \leqq b, x \geqq 0 \right\} = \max_{y \in \mathbb{R}^m} \left\{ b^\top y \colon A^\top y \leqq c, y \leqq 0 \right\}$$

have integer optimum solutions if both optima are finite.

2.3 Multi-Objective Optimization

This section extends single-objective optimization problems to *multi-objective* optimization problems and is mainly based on the textbooks by Steuer [1986] and Ehrgott [2005].

Multi-objective optimization (MOO) is gaining significant attention in academic literature due to its ability to address the conflicting objectives which appear in many real-world problems, including supply chains, transportation networks, environmental management, financial portfolio optimization and several others. In general, there is no solution that simultaneously optimizes all conflicting objectives. In a-posteriori approaches, one is interested in finding solutions with the property that none of the objectives can be improved without the deterioration of at least one other objective.

Definition 2.38. *A* multi-objective optimization (MOO) *problem is defined by*

$$\min_{x \in \mathcal{X}} f(x) = (f_1(x), \ldots, f_p(x))^\top, \tag{MOO}$$

where $f\colon \mathbb{R}^n \to \mathbb{R}^p$ *is a vector-valued* objective function *composed of* $p \geq 2$ *real valued objective functions* $f_k\colon \mathbb{R}^n \to \mathbb{R}$ *for* $k \in \{1, \ldots, p\}$ *and* $\mathcal{X} \subseteq \mathbb{R}^n$ *denotes the* set of feasible solutions. *We call* $\mathbb{R}^n$ *the* decision space *and* $\mathbb{R}^p$ *the objective space. The image of the feasible set*

$$\mathcal{Y} \coloneqq f(\mathcal{X}) = \{f(x)\colon x \in \mathcal{X}\} \subseteq \mathbb{R}^p$$

is called the set of feasible outcome vectors *in the objective space.*

Pareto Concept of Optimality We assume that the objective functions are in general conflicting, which implies that we exclude the existence of an *ideal* solution that minimizes all objectives simultaneously. In this situation, the *concept of Pareto optimality* is used, based on the component-wise order in $\mathbb{R}^p$.

Definition 2.39. *Let* $y^1, y^2 \in \mathbb{R}^p$. *We write:*

- $y^1 \leqq y^2$ *if* $y^1_i \leq y^2_i$ *for* $i = 1, \ldots, p$,
- $y^1 \leq y^2$ *if* $y^1 \leqq y^2$ *but* $y^1 \neq y^2$ *and*
- $y^1 < y^2$ *if* $y^1_i < y^2_i$ *for* $i = 1, \ldots, p$.

Definition 2.40. *A feasible solution* $x^* \in \mathcal{X}$ *is* efficient *if there is no feasible solution* $x \in \mathcal{X}$ *such that* $f(x) \leq f(x^*)$. *If* x^* *is efficient,* $f(x^*)$ *is called* nondominated point. *If for* $x, x' \in \mathcal{X}$ *hold that* $f(x') \leq f(x)$, *then* x' *is said to* dominate x *and* $f(x')$ dominates $f(x)$. *Feasible solutions* $x^*, x' \in \mathcal{X}$ *are* equivalent *if* $f(x^*) = f(x')$. *The set of efficient solutions is denoted by* $\mathcal{X}_E \subseteq \mathcal{X}$ *and the set of nondominated points by* $\mathcal{Y}_N \subseteq \mathcal{Y}$. *Moreover, a feasible solution* x' *is called weakly efficient if there is no other solution* x *such that* $f(x) < f(x')$. *Furthermore, a feasible solution* $x^* \in \mathcal{X}$ *is called properly efficient (in Geoffrion's sense, Geoffrion [1968]), if it is efficient, and if there is a real number* $M > 0$ *such that for all* k *and* $x \in \mathcal{X}$ *satisfying* $f_k(x) < f_k(x^*)$ *there exists an index* j *such that* $f_j(x^*) < f_j(x)$ *such that*

$$\frac{f_k(x^*) - f_k(x)}{f_j(x) - f_j(x^*)} \leq M.$$

The image $y^* = f(x^*)$ *is called properly nondominated.*

In the following, it is assumed that the multi-objective optimization problem has at least one nondominated point, i.e., $\mathcal{Y}_N \neq \emptyset$, and that $\mathcal{Y}$ is a closed set. The polyhedron $\mathcal{Y}^{\geqq} := \operatorname{conv}(\mathcal{Y}_N + \mathbb{R}^p_{\geqq})$ is called the *upper image* of $\mathcal{Y}$.

Minimal Complete Set When solving multi-objective problems, they can be solved concerning different solution concepts, and it is important to specify what class of set of efficient solutions is searched and be aware of the (partial or complete) enumeration of equivalent solutions. For example, the complete set of all efficient solutions $\mathcal{X}_E$ can be searched. However, if we speak about the task to determine all nondominated points $\mathcal{Y}_N$, we want to determine a minimal complete set of $\mathcal{X}' \subseteq \mathcal{X}_E$.

Definition 2.41. *If a subset $\mathcal{X}'_E \subseteq \mathcal{X}_E$ fulfills $f(\mathcal{X}'_E) = \mathcal{Y}_N$, i.e., for each $y \in \mathcal{Y}_N$ there exists at least one $x' \in \mathcal{X}'_E$ such that $f(x) = y$, $\mathcal{X}'$ is called a* complete set *of efficient solutions. A complete set is called* minimal complete set *if it contains no equivalent solutions, i.e., solutions mapping to the same nondominated point.*

Ideal and Nadir Point It is assumed that the objective functions are conflicting, implying that no solution minimizes all objectives simultaneously. Therefore, a point combining the minimal values for all objectives, called the *ideal point* (denoted by y^I), does not belong to $\mathcal{Y}_N$ but can be used as a component-wise lower bound for all nondominated points. We assume, without loss of generality, that the ideal point is positive, ensuring that the origin dominates it. Conversely, vectors that define upper bounds on objective function vectors y are represented by the *Nadir point* y^N, which is composed of the component-wise $y_i^N = \max\{y_i \colon y \in \mathcal{Y}_N\}$ over the nondominated set, assuming that $\mathcal{Y}_N$ is bounded.

Canonical Decision Problem There exist results on the **NP**-hardness of decision problems associated with multi-objective optimization problems.

Definition 2.42. *For a multi-objective optimization Problem MOO, the canonical decision problem MOO^{Dec} is defined as: Given a vector $k \in \mathbb{R}^p$ does there exists a feasible solution $x \in \mathcal{X}$ such that $f(x) \leqq k$.*

It is shown that MOO^{Dec} is **NP**-hard for general MOO problems, which can be seen as an informal evidence that MOO is a hard problem, while also being intractable in general.

Scalarization To solve multi-objective problems, scalarization based methods are a common concept in multi-objective optimization. It involves converting a multi-objective optimization problem, particularly its vector-valued objective function, into a sequence of optimization problems with scalar-valued objective function and/or parameterized constraints. Various scalarizations exist, which mainly differ in the complexity of the reformulation and the quality of the obtained solution. Ideally, the solution of a scalarization yields an efficient solution or is infeasible.

One straightforward scalarization method combines the multiple objectives into one by assigning a weighting parameter to each original objective and by summing these weighted objectives. This technique, known as the weighted-sum method, was introduced by Gass and Saaty [1955] as a "parametric function" for linear optimization problems with two objectives.

Definition 2.43. *Let $\|x\|_1$ denote the 1-norm of $x \in \mathbb{R}^p$, i. e., $\|x\|_1 \coloneqq \sum_{i=1}^p |x_i|$. For the weighted sum method, the set of* normalized weighting vectors *is defined as the set $\Lambda_p = \{\lambda \in \mathbb{R}^p_> \colon \|\lambda\|_1 = 1\}$ or $\Lambda_p^0 = \{\lambda \in \mathbb{R}^p_\geq \colon \|\lambda\|_1 = 1\}$ if weights equal to zero are included. The* weighted-sum scalarization *with $\lambda \in \Lambda_p$ or $\lambda \in \Lambda_p^0$ is defined as the parametric program*

$$P_\lambda \coloneqq \min_{x \in \mathcal{X}} \lambda^\top f(x) \tag{P_λ}$$

with $f(x)$ as in Definition 2.38.

Theorem 2.44 (Ehrgott 2005)**.** *For $\lambda \in \Lambda_p^0$, every optimal solution of P_λ is a weakly efficient solution of* (MOO)*. Moreover, every optimal solution of P_λ problem is (properly) efficient for* (MOO) *if $\lambda \in \Lambda_p$.*

Multi-Objective Linear Programming A specific, well-studied class of MOO are the *multi-objective linear programs (MOLP)* and *multi-objective integer linear programs* (MOILP).

Definition 2.45. *A* multi-objective linear optimization program (MOLP) *is defined by*

$$\min_{x \in \mathcal{X}} y(x) = (y_1(x), \dots, y_p(x))^\top = Cx, \tag{MOLP}$$

with $\mathcal{X} \coloneqq \{x \in \mathbb{R}^n \colon Ax \leqq b, x \geqq 0\}$. Thereby, the rows c^k $(k = 1, \dots, p)$ of the cost matrix $C \in \mathbb{R}^{p \times n}$ contain the coefficients of the p linear objective functions $y_k(x) = c^k x$ for $k \in \{1, \dots, p\}$ and the matrix $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$ formulates the m linear constraints.

If all decision variables are further restricted to the set of integers, i.e., $x \in \mathbb{Z}^n$, we obtain a multi-objective *integer* linear program. Note that if for a MOILP with $C \in \mathbb{Z}^{p \times n}$ or MOCO, we sometimes write $z(x) = Cx$ for the objective function.

Theorem 2.46 (Isermann 1974)**.** *Considering MOLP problems, for every efficient solution $\bar{x} \in \mathcal{X}_E$, there exists a $\lambda \in \Lambda_p$ such that $\bar{x}$ is an optimal solution of the weighted sum scalarization (P_λ).*

Hence, for a MOLP, the efficient set $\mathcal{X}_E$ equals the union over all sets of optimal solutions of P_λ for $\lambda \in \Lambda_p$.

However, in the non-linear case and for multi-objective combinatorial (MOCO) problems, nondominated points can also be located in the interior of the upper image, i.e., in $\operatorname{int}(\operatorname{conv}(\mathcal{Y} + \mathbb{R}^p_{\geqq}))$. These so-called *nonsupported points* typically outnumber these points that can be obtained with the weighted sum method [Visée et al., 1998]. To obtain these points, alternative scalarization methods, such as the ε-constraint method, are required. However, these approaches may disrupt the combinatorial structure. Consequently, determining the set of efficient solutions for MOILP is significantly more challenging as compared to MOLP, and finding the complete nondominated point set is intractable for most MOILP problems. Current methods struggle to scale with the number of objectives [Ehrgott, 2005]. Therefore, it is important to distinguish between different types of efficient solutions. This distinction will be examined in the subsequent section, focusing on the major problem class in this thesis, known as multi-objective combinatorial optimization (MOCO).

2.3.1 Multi-Objective Combinatorial Optimization

Formally, a single-objective combinatorial optimization problem (with a linear objective function) can be formulated as follows:

Given: finite ground set E
feasible subsets $\mathcal{I} \subseteq 2^E$
$c : E \to \mathbb{R}$
for $F \subseteq E : c(F) := \sum_{e \in F} c(e)$
Find: $I^* \in \mathcal{I}$ such that $c(I^*) \leq c(I)$ for all $I \in \mathcal{I}$

For example, E can be the set of edges of a connected undirected graph, and $\mathcal{I}$ can be the set of all spanning trees in this graph. (Graphs and spanning trees will be presented in Section 2.4). However, in the following definition of a multi-objective combinatorial optimization problem we use the *index formulation* of a combinatorial problem.

Definition 2.47. *A* multi-objective combinatorial optimization *(MOCO) problem is defined as*

$$\min_{x \in \mathcal{X}} z(x) = (z_1(x), \ldots, z_p(x))^\top = Cx, \tag{MOCO}$$

where $\mathcal{X} := \{x \in \{0,1\}^n : Ax = b\}$*, with the cost matrix* $C \in \mathbb{Z}^{p \times n}$ *contains the rows* c^k *of coefficients of* p *linear objective functions* $z_k(x) = c^k x$ *for* $k \in \{1, \ldots, p\}$ *and* $A \in \mathbb{Z}^{m \times n}, b \in \mathbb{Z}^m$ *describe the* m *constraints. The constraints define combinatorial structures such as paths, trees, or cycles in a network or partitions of a set. All coefficients are assumed to be integers.*

This definition includes various problems such as multi-objective spanning trees, shortest paths, knapsack, and assignment problems.

Nondominated Frontier For MOLP or MOCO problems, the *nondominated frontier* can be defined in the following way:

Definition 2.48. *The* nondominated frontier *is the set*

$$\{y \in \operatorname{conv}(\mathcal{Y}_N) : \operatorname{conv}(\mathcal{Y}_N) \cap (y - \mathbb{R}^p_{\geqq}) = \{y\}\}.$$

The nondominated frontier equals the nondominated set of $\operatorname{conv}(\mathcal{Y}_N)$ and can be characterized as the union of the *maximal nondominated faces* of the upper image $\mathcal{Y}^{\geqq}$ [Ehrgott, 2005]. Hereby, a *nondominated face* $F \subseteq \mathcal{Y}^{\geqq}$ is a face of the upper image $\mathcal{Y}^{\geqq}$ such that all its points are nondominated with respect to $\mathcal{Y}^{\geqq}$. Similarly, $F \subseteq \mathcal{Y}^{\geqq}$ is called a *weakly nondominated face* if all its points are weakly non-dominated. A face F is called *maximally nondominated* if there is no other nondominated face G of $\mathcal{Y}^{\geqq}$ such that F is a proper subset of G ($F \subsetneq G$). This would imply that the dimension of G is be greater than the dimension of F. The preimage $F_{\mathcal{X}}$ of a maximally nondominated face $F_{\mathcal{Y}}$ of $\mathcal{Y}^{\geqq}$, i.e., all solutions whose image lies in $F_{\mathcal{Y}}$, is denoted as the *maximally efficient face*. Note that $F_{\mathcal{X}}$ might not be a face of the feasible set, the polyhedron $\mathcal{X}$, since multiple feasible solutions (located on different faces of $\mathcal{X}$) could map to the same nondominated point $y \in \mathcal{Y}$. Figure 2.1 illustrates the nondominated frontier, the weighted sum method, the Ideal and Nadir point of a bi-objective problem ($p = 2$).

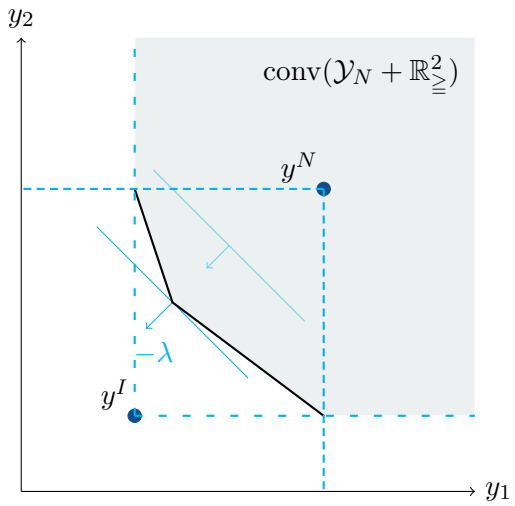


Figure 2.1: An illustration of the outcome space of a bi-objective problem, highlighting the concept of the nondominated frontier, Ideal and Nadir point. The black lines display the complete nondominated frontier and can be found when varying λ within Λ^p.

(Extreme) Supported and Nonsupported Solution Two major challenges are associated with MOCO problems as compared to their single objective counterparts. Firstly, they belong to the class of computationally intractable problems [Ehrgott, 2005]. Secondly, in MOCO or MOILP problems, nondominated points can be located in the interior of the upper image, which typically outnumber the solutions on the nondominated frontier. (see, e.g., Visée et al. 1998). Hence, a distinction between different classes of efficient solutions is required.

Definition 2.49. *The set of efficient solutions $\mathcal{X}_E$ can be distinguished in the following classes:*

1. Extreme supported solutions *are those solutions whose image lies on the vertex set of the upper image. Their image is called an* extreme supported nondominated point*; we use the notations $\mathcal{X}_{ES}$ and $\mathcal{Y}_{ES}$ for the extreme supported efficient solution set and extreme supported nondominated point set, respectively.*

2. *An efficient solution is called* supported efficient solution *if it is an optimal solution to the weighted sum scalarization P_λ for $\lambda \in \Lambda_p$, i.e., an optimal solution to a single objective weighted-sum problem where the weights are strictly positive. Its image is called* supported nondominated point*; we use the notations $\mathcal{X}_S$ and $\mathcal{Y}_S$ for the supported efficient solution set and supported nondominated point set, respectively. A supported nondominated point is located on the* nondominated frontier*, i.e., located on the union of the* maximal nondominated faces*.*

3. Weakly supported efficient solutions *are efficient solutions that are optimal solutions of P_λ for $\lambda \in \Lambda_p^0$, i.e., an optimal solution to a single-objective weighted sum problem with weights strictly or equal to zero. Their images in the objective space are*

weakly supported nondominated points. *All weakly supported nondominated points are located on the boundary of the* upper image $\mathcal{Y}^{\geqq}$.

4. Nonsupported efficient solutions *are efficient solutions that are not optimal solutions of* P_λ *for any* $\lambda \in \Lambda_p^0$. Nonsupported nondominated points *lie in the interior of the upper image.*

Figure 2.2 illustrates extreme supported, supported, and nonsupported nondominated points as well as the upper image in the bi-objective case.

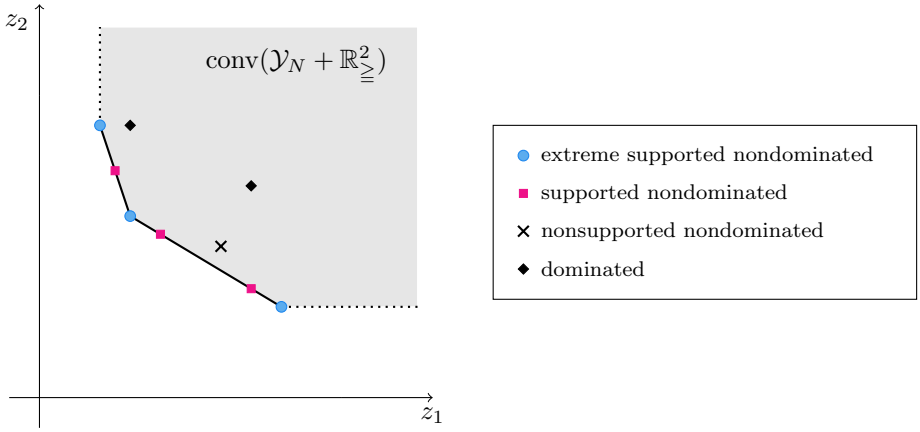


Figure 2.2: Illustration of the upper image $\mathcal{Y}^{\geqq} := \text{conv}(\mathcal{Y}_N + \mathbb{R}^2_{\geqq})$ and the different solution types.

The nonsupported nondominated points typically outnumber the supported ones and are not as straightforward to obtain as the supported nondominated points. However, as observed in Sayın [2024], the supported nondominated set or extreme supported nondominated set already provide high-quality representations.

However, in the context of MOCO problems, other definitions and characterizations for supportedness are also used. Chapter 4 introduces these different definitions and proves that these are not equivalent in the case of general optimization problems. As a result, they generate different sets of supported efficient solutions and, consequently, different supported nondominated point sets with different properties. It also formally introduces the distinction between supported and weakly supported nondominated points and proves that the weakly supported nondominated points can be a proper subset of supported nondominated points in MOCO problems.

Solution Methods Two highly promising approaches for solving MOCO problems are *generic scalarization-based algorithms* and *two-phase methods*. A comprehensive review of exact methods can be found in Ehrgott and Wiecek [2005] and with a specific focus on discrete MOO in Halffmann et al. [2022].

Generic scalarization-based methods decompose the overall problem into a series of scalarized single-objective problems that can be solved with efficient available single-objective IP solvers. The scalarization transforms the optimization problem with multiple

objectives into a problem with a single objective and parameters that somehow control how the original objectives are embedded into the transformed problem. The independence from specific problem structures is appealing since the transformation done during the scalarization typically destroys or alters the combinatorial structure of the problem. Generic approaches have many different implementation possibilities and offer flexibility, including choices of the sequence of subproblems, scalarization strategies, and decisions regarding the computation of the complete or a representation of the set of nondominated points. Generic scalarization-based methods and their efficient implementation possibilities will be presented in Chapter 3.

Two-phase methods represent another well-known approach. The first phase determines the set of (extreme) supported nondominated points by using that the image of these points are optimal solutions of weighted sum problems with positive weights. In the second phase, other efficient solutions, such as the nonsupported or, if necessary, the remaining nonextreme supported, are determined, usually using enumerative methods. In that phase, information from supported nondominated points from the first phase is used to determine a search area in the objective space that contains all nonsupported nondominated points. Several established algorithms are used for both phases for bi-objective problems. Commonly, the first phase is done using *dichotomic search* [Aneja and Nair, 1979], while the second phase relies on enumerative methods, such as ranking approaches (see, e.g., Przybylski et al. [2008]) and branch and bound procedures (see, e.g.,Visée et al. [1998]). A survey of multi-objective branch and bound methods is given by Przybylski and Gandibleux [2017] and recently in Bauß et al. [2024].

However, methods for enumerating the extreme supported nondominated points in MOCO problems with more than two objectives are rather limited. Przybylski et al. [2010a] present a recursive method for identifying these points in general MOILP problems. Özpeynirci and Köksalan [2010] and Przybylski et al. [2019] propose a method based on a dichotomic approach to compute this set. Additionally, weight space decomposition methods, such as those used in multi-objective simplex variants (e.g., Yuf and Zeleny [1976]), offer alternative approaches. For a broader discussion of these and other methods, we again refer to the surveys in Ehrgott and Wiecek [2005] and Halffmann et al. [2022]. Another important method for enumerating extreme supported nondominated points is the *dual variant of the Benson algorithm* [Ehrgott et al., 2012]. A key proof concerning the running time of the dual variant is presented by Bökler and Mutzel [2015], which plays an important role in the following section.

Multi-objective Problem as Enumeration Problem MOCO problems, like any discrete multi-objective optimization problem, can be considered as enumeration problems. This involves enumerating all nondominated points or subsets of the nondominated point set with respect to other solution concepts.

Optimization problems are often classified based on whether they can be solved in polynomial or exponential time. However, for many MOCO problems, the cardinality of $\mathcal{X}_E$ and $\mathcal{Y}_N$ itself grows exponentially with respect to the input. Thus, even if it is an easy task to determine one efficient solution or nondominated point, enumerating all desired

solutions in polynomial time is impossible. Hence, we are interested in whether such problems are solvable in output-polynomial time. A MOCO problem is called intractable if $|\mathcal{Y}_N|$ can be exponential in the size of an input instance.

Since output-polynomial time is defined with respect to both the input and the output of an algorithm, it is crucial to specify the output being considered. An algorithm for solving MOO might be output-polynomial when the output is defined as the set of all efficient solutions. However, the same runtime may not be output-polynomial if the output is instead defined as the set of all nondominated points.

To formalize this, let C be the configuration set of all efficient solutions, and C^* be the configuration set of all nondominated points. An output-polynomial time algorithm $E = (I, C)$, which enumerates all efficient solutions exactly once, will also yield all nondominated points. However, this does not guarantee that the algorithm is output-polynomial with respect to the configuration set of all nondominated points C^*, since there may exist exponentially many efficient solutions that map to only a few nondominated points. Consequently, the same algorithm, when considered with the configuration set C^* as $E = (I, C^*)$, may output elements of the configuration set (i.e., the nondominated points) multiple times, potentially even an exponential number of times. As illustrated in Section 2.5.3, there exists an example of a multi-objective flow problem that contains $\binom{2n-1}{n}$ efficient solutions, all mapping to a single nondominated point.

Some results show under which circumstances a given set can either be found in output-polynomial time or not. For enumerating the extreme supported nondominated points in MOCO problems, the following result could be shown by Bökler and Mutzel [2015].

Theorem 2.50 (Bökler and Mutzel, 2015)**.** *For every MOCO problem P with a fixed number of objectives, the set of extreme nondominated points of P can be enumerated in output-polynomial time if the weighted-sum scalarization of P can be solved in polynomial time.*

For any MOCO problem satisfying the required conditions, the dual variant of the Benson Algorithm [Bökler and Mutzel, 2015] can be employed to enumerate the set of extreme supported nondominated points in output-polynomial time. Throughout this thesis, similar characterizations for solving MOCO with respect to other solution concepts, such as (extreme) supported efficient solutions or nondominated points for MOCO problems, will be presented.

2.4 Graph and Network

This section introduces several basic definitions of graph theory based on the textbooks by Ahuja et al. [1993] and Diestel [2006]. Additionally, some of the most significant graph problems considered throughout the thesis are presented. The section begins by defining undirected and directed graphs.

2.4.1 Undirected Graph

Definition 2.51. *An* (undirected) graph *is an ordered pair* $G = (V, E)$*, where* V *is the set of* n nodes *and* $E = \{\{i, j\} : i, j \in V\}$ *is the set of* m *(unordered) pairs of* V *called* edges*. For an edge* $\{i, j\} \in E$*, it is allowed that* $i = j$*. In this case, the edge* $\{i, i\}$ *is called a* loop*. Furthermore, the set* E *can also contain multiple edges between the same pairs of nodes, known as* parallel edges*. A graph is called* simple *if neither parallel edges nor loops exist. Otherwise, we refer to such graphs as* multigraphs*, where a multi-set technically replaces the set* E *to accommodate multiple occurrences of the same edge.*

A graph G is called *finite* if $|V|$ and $|E|$ are finite. In the following, we assume that the considered graphs are finite and simple.

Given a mapping $\phi\colon E \to Y$ that assigns each edge $e \in E$ a value in an arbitrary set Y, the image of an edge $\{i, j\}$ under ϕ is denoted as $\phi_{ij} = \phi(\{i, j\})$. Similarly, for a mapping $\gamma\colon V \to Y'$, the image of node i under γ is denoted as $\gamma_i = \gamma(i)$.

Adjacency and Incidence Two nodes i and j are *adjacent* if $\{i, j\} \in E$. In this case, j is called a *neighbor* of i. A node $i \in V$ is *incident* to an edge $e \in E$ if $i \in e$. Two edges $e, f \in E$ are *adjacent* if $e \cap f \neq \emptyset$. Otherwise, they are called *disjoint*. The graph's structure can be represented through its *adjacency matrix* or its *incidence matrix*. The *adjacency matrix* of a graph $G = (V, E)$ is the $|V| \times |V|$ matrix M^{AD}, where m_{ij}^{AD} is defined as the number of edges connecting i and j. In a simple graph, this matrix indicates the existence of the edge $(i, j) \in E$ if $m_{ij}^{AD} = 1$ and 0 otherwise. Note that for an undirected graph, the adjacency matrix is symmetric, i.e., $m_{ij}^{AD} = m_{ji}^{AD}$.

The *incidence matrix* of a graph$G = (V, E)$ is the $|V| \times |E|$ matrix M^{IN}, where $m_{i,j}^{IN}$ represents the relationship between node i and edge e. In an undirected graph, $m_{ie}^{IN} = 1$ if $i \in V$ is incident to edge $e \in E$ and 0 otherwise. Each column of the incidence matrix has exactly two ones corresponding to the two nodes connected by the edge.

Degree and Neighbor The set of incident edges of a node i is defined by $\delta(i) = \{\{i, j\} : \{i, j\} \in E\}$. The *degree* of a node $i \in V$, denoted by $\deg(i)$, is the number of its incident edges, i.e., $\deg(i) = |\delta(i)|$. The set of *neighbors* of the node $i \in V$ is denoted by $N(i) = \{j \in V : \{i, j\} \in E\}$.

Walk, Path, and Cycle A *walk* in an undirected graph $G = (V, E)$ is a finite sequence $P = \{v_0, e_1, v_1, \ldots, e_k, v_k\}$ with $k \geq 0$, where v_0 is the start node and v_k is the end node, and $e_i = \{v_{i-1}, v_i\} \in E$. This is a walk from v_0 to v_k, briefly called a (v_0, v_k)-walk. In simple graphs, we also write $P = \{v_0, v_1, \ldots, v_k\}$ instead of $P = \{v_0, a_1, v_1, \ldots, a_k, v_k\}$. The length $|P|$ of a walk is given by the number of its k edges. A *node-disjoint walk* is a walk in which all nodes are pairwise distinct, called a *path*. Therefore, a path is a walk without any repetition of nodes. A *cycle* is a walk $\{v_0, \ldots, v_k\}$ for which $v_0 = v_k$ holds, and the nodes $v_1, \ldots, v_k$ are pairwise distinct. A graph G is called *acyclic* if it contains no cycles.

Cut For a graph G and a node subset $X \subset V$, $\delta(X) = E(X, V\backslash X) = \{\{i,j\} \in E : i \in X, j \in V\backslash X\}$ is called a *cut* in G and $\delta(X)$ is the *induced cut* of X. Thus, a cut partitions the node set into two parts and defines a set of edges consisting of those edges that have one endpoint in X and another endpoint in $V\backslash X$. If for two distinguished nodes s and t it holds that $s \in X$ and $t \notin X$, $\delta(X)$ is called a s-t cut.

Subgraph A *subgraph* of a graph $G = (V, E)$ is a graph $G' = (V', E')$ such that $V' \subseteq V$ and $E' \subseteq \{\{i,j\} \in E : i,j \in V\}$. The subgraph G' is *induced* by V' if $E' = \{\{i,j\} \in E : i,j \in V'\}$, meaning it contains each edge in E having both endpoints $i,j \in V'$. The induced subgraph is denoted as $G(V')$. Note that $E(V')$ denotes the set of edges that have both endpoints in V', i.e., $e = (i,j) \in E(V')$ if $i,j \in V'$. A graph $G' = (V', E')$ is a *spanning subgraph* of $G = (V, E)$ if G' is a subset and $V' = V$.

Connectivity and Tree Spanning subgraphs are useful for studying the properties and structures of the original graph while considering only a subset of its edges. An important type of a spanning subgraph is the *spanning tree*. A tree is an important concept in graph theory that arises in various network flow algorithms and properties discussed in this thesis. In our subsequent discussion in later chapters, we use some of the following elementary properties of trees.

Two vertices $i,j \in V$ are called *connected* if a path exists from i to j. A graph is called *connected* if every pair of nodes is connected. An acyclic graph is denoted as a *forest*. If a forest is connected, it is a *tree*. A tree T is a *spanning tree* of G if T is a spanning subgraph of G. Each spanning tree of a connected graph contains exactly $n-1$ edges. The edges $e \in E(T)$ are *tree edges*, and edges $e \in E\backslash E(T)$ are called *non-tree edges*.

Property 2.52 (Ahuja et al., 1993)**.** *For a tree of G, the following conditions hold:*

- *A tree with n nodes contains exactly $n-1$ edges.*
- *A tree has at least two leaf nodes (i.e., nodes with degree 1).*
- *A unique path connects every two nodes of a tree.*

A connected subgraph of a tree is called *subtree*. A tree can have an arbitrary but fixed node called its *root*. Then we regard a *rooted tree* as though it was hanging from its root.

Let T be a spanning tree of the graph G. Adding any non-tree edge to the spanning tree T creates exactly one cycle. Any such cycle is referred to as a *fundamental cycle* or *induced cycle* of G with respect to the tree T. Since a simple graph with $|E| = m$ contains $m-n+1$ non-tree arcs, there are $m-n+1$ induced cycles with respect to any spanning tree. A spanning tree is again obtained if one arbitrary arc in an induced cycle is deleted. This property allows the creation of different trees by replacing one tree arc with a non-tree arc in an initial tree to obtain a new tree.

2.4.2 Directed Graph

A *directed graph* is defined in the same manner as an undirected graph, except that edges are ordered pairs of distinct nodes, referred to as arcs.

Definition 2.53. *A* directed graph *or* digraph *is a pair $D = (V, A)$, where $A \subseteq V \times V$ is a set of m ordered pairs of V. An element $(i, j) \in A$ denotes a* directed *or* oriented arc *from i (tail) to j (head). Arcs (i, j) and (j, i) are referred to as* anti-parallel*, and (j, i) is called the* reverse arc *of (i, j). A* directed network *is a directed graph where nodes and arcs have associated numerical values (typically, costs, capacities, supplies and demands).*

In slight abuse of the notation, we use the terms "graph" and "network" synonymously.

Definition 2.54. *For each directed graph D, an* underlying undirected graph $G_D = (V, E)$ *is defined by neglecting the orientation of the arcs in A, i.e., $E := \{\{i, j\} : (i, j) \in A\}$.*

Note that some definitions applicable for undirected graphs in the previous subsection can be analogously applied to directed graphs or are transferable from $D = (V, A)$ when considering the underlying undirected graph $G_D = (V, E)$ and are not repeated in this section.

Adjacency and Incidence The *adjacency matrix* of a directed graph is the $|V| \times |V|$ matrix, where m_{ij}^{AD} is the number of arcs $(i, j) \in A$. In a simple graph, this matrix indicates the presence or absence of an arc between nodes i and j and $m_{ij}^{AD} = 1$ if $(i, j) \in A$ and 0 otherwise. Unlike the undirected case, this matrix is not necessarily symmetric for directed graphs due to the ordering of the arc (i, j) in the directed case.

The *incidence matrix* of a directed graph $D = (V, A)$ is the $|V| \times |A|$ matrix M^{IN}, where $m_{ia}^{IN} = 1$ if $i \in V$ is the tail of arc $a = (i, j) \in A$, and $m_{ja}^{IN} = -1$ if node j is the head of a, and 0 otherwise. Each column of the incidence matrix has exactly one 1 and one -1, corresponding to the direction of the arc from one node to another.

In-degree and Out-degree The set $N^{in}(i) = \{j \in V : (j, i) \in A\}$ are the *predecessors* and $N^{out}(i) = \{j \in V : (i, j) \in A\}$ *successors* of i, respectively. The set of neighbors $N(i)$ of i is $N(i) = N^{in}(i) \cup N^{out}(i)$. The arc sets $\delta^{in}(i) = \{(j, i) : (j, i) \in A\}$ and $\delta^{out}(v) = \{(i, j) : (i, j) \in A\}$ are the *incoming* and *outgoing* arcs of node i, respectively. Again, $\delta(i) = \delta^{in}(i) \cup \delta^{out}(i)$. The cardinality of the set of incoming arcs of a node i is known as its *in-degree*, denoted by $\deg^{in}(i) = |\delta^{in}(i)|$, and the cardinality of the set of outgoing arcs of a node i is known as its *out-degree*, denoted by $\deg^{out}(i) = |\delta^{out}(i)|$.

Directed Walk, Path and Cylce A *(directed) walk* in a digraph $D = (V, A)$ is a finite sequence $\{v_0, a_1, v_1, \ldots, a_k, v_k\}$, where $(v_{i-1}, v_i) \in A$. If all nodes in the walk are distinct, it is called a *(directed) path*. A *(directed) cycle* is a directed walk where $v_0 = v_k$ and $v_1, \ldots, v_k$ are pairwise distinct. The set of arcs of an undirected walk, path, or cycle in a digraph, i.e., a walk, path, cycle in the undirected underlying graph, can be partitioned into forward and backward arcs. An arc $(i, j) \in A$ in the undirected walk is a *forward arc* if the walk visits node i prior to node j, and is a backward arc otherwise.

Cut and Connectivity A directed graph is called *strongly connected* if the graph contains at least one directed path from every node to every other node. In the directed case, the induced cut is defined similarly to the undirected case but separated into $\delta^{out}(X) = A(X, V\backslash X) = \{(i,j) \in A : i \in X, j \in V\backslash X\}$ and $\delta^{in}(X) = A(V\backslash X, X) = \{(i,j) \in A : i \in V\backslash X, j \in X\}$. It holds that $\delta^{out}(X) = \delta^{in}(V\backslash X)$.

2.4.3 Directed Depth First Search

The following presents a graph-searching algorithm for a directed graph. Graph searching involves systematically following the edges or arcs to visit the nodes. This process can reveal much about the graph's structure, and many algorithms begin by searching their input graph to gather this structural information. Specifically, the directed depth-first search algorithm is a crucial foundation for a later algorithm introduced in Chapter 6. The reader is assumed to be familiar with undirected depth-first search, which is covered in standard literature such as Ahuja et al. [1993]. This subsection is based on Cormen et al. [2001].

Basic Principles The depth-first search (DFS) algorithm explores a graph D in $\mathcal{O}(m+n)$ time by following a strategy of exploring "deeper" into the graph whenever possible, i.e., leaving nodes as early as possible. It explores arcs from the most recently discovered node with unexplored arcs. Once all arcs from this node have been explored, DFS backtracks to the preceding node to continue the search. This process ends if all reachable nodes from the source are discovered. If undiscovered nodes remain, the DFS procedure will select one node as a new source and repeat the search. In that case, DFS will create more than one DFS tree. This process continues until every node in the graph has been discovered.

The correctness and complexity analysis will not be covered in the following description and can be found in Cormen et al. [2001]. However, due to the significance of different arc types in a DFS forest, the (directed) DFS algorithm, the arc types, and the key properties of the DFS forest are presented.

DFS Forest Because the search may repeat from multiple sources, the DFS algorithm creates a DFS-forest, denoted by $\Gamma = \{T_1, \dots T_l\}$ of D, consisting of several *depth-first trees*. For each tree $T \in \Gamma$ and each node $i \in T$, π_i is defined as the predecessor of $i \in T$. Furthermore, let dfs(i) denote the DFS number (analogous to discovery time in Cormen et al. [2001]) of node i, i.e., the number when node i is first visited (discovered) during the DFS process. Let further t_i denote the time when node i is *finished*, that is, when all arcs of node i are considered. These timestamps for each node i provide important information about the graph structure. For a tree $T \in \Gamma$, the node i with the lowest dfs-number is called the *root*, and the tree T is called *rooted tree*. A rooted tree is regarded as though it were hanging from its root, and the arcs in a rooted tree define predecessor-successor (or parent-child) relationships. A node $j \in T$ with $T \in \Gamma$ is called *ancestor* of $i \in T$ if j is in the path from the root to the node only using arcs in the tree. A node j is called *descendant* of node i in T if and only if i is an ancestor of node j. Given two nodes $i, j \in T$, their *lowest common ancestor* is the deepest node that is an ancestor of both nodes. This

means it is the node with highest DFS number that has both nodes as descendants (where we allow a node to be a descendant of itself).

Types of Arcs in DFS The DFS-forest contains different arc types, which will be presented in the following.

Definition 2.55 (Cormen et al., 2001)**.** Tree arcs *are the arcs in the trees of the DFS-forest.* Backward arcs *are the arcs* (i, j) *connecting a vertex* i *to an ancestor* j *in the same DFS-tree. Non-tree arcs* (i, j) *are called* forward arcs *if the arc connects a vertex* i *to a descendant* j *in the same DFS-tree. All other arcs are called* cross arcs*.*

Algorithm: Directed DFS with Arc Type Detection In the context of a DFS, nodes are typically marked with three different colors to track their state during the traversal. Note that sometimes numbers instead of colors are used. A node i is *white* if it has not been discovered yet, i.e., before time $\text{dfs}(i)$. Initially, all nodes are white. Gray if it has been discovered but not fully explored. This means that the DFS has started visiting this node and is in the process of visiting its descendants, i.e., in between time $\text{dfs}(i)$ and t_i and black if it has been fully explored, i.e., in time after t_i. The pseudocode in Algorithm 1 and Algorithm 2 displays the basic DFS algorithm, which directly detects the different types of arcs in the DFS forest.

Algorithm 1: Directed DFS with Arc Type Detection

Input: Directed graph $D = (V, A)$
Output: DFS numbers $\text{dfs}(i)$ for $i \in V$, predecessors π_i for $i \in V$, arc types

```
for each node i ∈ V do
    color(i) ← white
    π_i ← ∅
    dfs(i) ← 0
end
time ← 1
for each node i ∈ V do
    if color(i) = white then
        DFS-Visit(i,time)
    end
end
```

2.4.4 Multi-Objective Graph Problems

This section presents two fundamental graph problems that are relevant for the thesis. Both problems have numerous applications in optimization.

Multi-Objective Minimum Spanning Tree The minimum spanning tree (MST) problem is a well-studied problem in combinatorial optimization. It involves finding a spanning tree

Algorithm 2: DFS-Visit

```
1  procedure DFS-Visit(i,time):
2      color(i) ← gray
3      dfs(i) ← time
4      time ← time +1
5      for each node j adjacent to i do
6          if color(j) = white then
7              arc (i, j) is a tree arc
8              DFS-Visit(j)
9          end
10         else if color(j) = gray then
11             arc (i, j) is a back arc
12         end
13         else if color(j) = black then
14             if dfs(i) < dfs(j) then
15                 arc (i, j) is a forward arc
16             end
17             else
18                 arc (i, j) is a cross arc
19             end
20         end
21     end
22     color(i) ← black
23 end
```

of a weighted, connected, undirected graph that minimizes the sum of its edge weights. For a comprehensive overview, we refer to the textbook by Ahuja et al. [1993]. For the multi-objective variant, we refer to Ehrgott et al. [2012].

Definition 2.56. *Given a Graph $G = (V, E)$ and a cost function $c\colon E \to \mathbb{R}^p$ that assigns a cost vector to each edge, the* multi-objective minimum spanning tree *problem is defined as*

$$\min_{T \in \mathcal{T}} c(T) = (c_1(T), \ldots, c_p(T))^\top = Cx, \tag{MMST}$$

where $\mathcal{T}$ is the set of all spanning trees of G, $C \in \mathbb{R}^{p \times n}$ is the cost matrix containing the rows c^k of coefficients of the p linear objective functions and $c_i(T) = \sum_{e \in T} c_e^i$.

The single objective ($p = 1$) problem can be solved in polynomial time with the well-known greedy algorithms such as Prim's [Prim, 1957] and Kruskal's [Kruskal, 1956], which can be implemented to run $\mathcal{O}(m + n \log n)$ time; see, e.g., Schrijver [2003]. The multi-objective minimum spanning tree problem can also be formulated as the following multi-objective integer linear program:

$$
\begin{aligned}
\min \quad & Cx \\
\text{s.t.} \quad & \sum_{e \in E} x_e = n - 1 \\
& \sum_{e \in E[U]} x_e \leq |U| - 1 && \forall\, U \subseteq V, S \neq \emptyset \\
& x_e \in \{0, 1\} && \forall\, e \in E
\end{aligned}
$$

where the cost matrix $C \in \mathbb{Z}^{p \times n}$ contains the rows c^k of coefficients of the p linear objective functions and where x_e is a binary variable indicating whether edge e is included in the MST ($x_e = 1$) or not ($x_e = 0$) and $E[U] \coloneqq \{e = (i, j) \in E \colon i, j \in U\}$.

Any feasible solution x defines an incidence vector of a spanning tree of G. The convex hull of the incidence vectors of all spanning trees of G is called the *spanning tree polytope*, which has an integer polytope [Schrijver, 2003]. Therefore, we can replace $x \in \{0, 1\}^m$ by $x_e \geq 0$ for all $e \in E$ and consider the linear relaxation problem to obtain an optimal solution. Note that this so-called *subtour elimination* IP formulation uses exponentially many constraints.

The multi-objective minimum spanning tree is known to be intractable and the finished decision problem and the canonical decision problem are known to be **NP**-hard, even in the case of two objectives.

Theorem 2.57 (Camerini et al., 1984)**.** *The finished decision problem of the bi-objective spanning tree problem is* **NP***-complete.*

The **NP**-completeness proof is done by a polynomial time reduction from the Knapsack Problem to the finished BMST (BMST^{FIN}) problem.

Theorem 2.58 (Hamacher and Ruhe, 1994)**.** *The bi-objective spanning tree problem is intractable.*

Proof. Consider the complete graph on n nodes, i.e., G contains n nodes and $m = n(n-1)/2$ edges, connecting every pair of distinct nodes by a unique edge. This graph contains n^{n-2} spanning trees. For each edge e_i, define $c^1_{e_i} = 2^{i-1}, c^2_{e_i} = 2^m - 2^{i-1}$ which implies $c^1_{e_i} + c^2_{e_i} = 2^m$. Therefore, for all spanning trees $T \in \mathcal{T}$, the corresponding outcome vectors are located on the hyperplane with $c^1(T) + c^2(T) = (n-1)2^m$. Additionally, $c^1(T_1) \neq c^1(T_2)$ for all pairs of spanning trees $T_1, T_2 \in \mathcal{T}$ with $T_1 \neq T_2$. As a result, all spanning trees have pairwise incomparable weights and are thus efficient. With all weights being different, $|\mathcal{Y}_N| = |\mathcal{T}| = n^{n-2}$. □

Multi-Objective Shortest Path Problem The shortest path problem involves finding the shortest path between two nodes in a weighted, directed graph, where the sum of the arc weights along the path is minimized. For a comprehensive overview, we refer again to the textbook by Ahuja et al. [1993]. For the multi-objective variant, we refer to Ehrgott et al. [2012].

Definition 2.59. *Given a directed graph $D = (V, A)$, a cost function $c\colon A \to \mathbb{R}^p$ that assigns a cost vector to each arc, a source node $s \in V$, and a target node $t \in V$, the (multi-objective)* shortest path *problem is defined as:*

$$\min_{P \in \mathcal{P}_{s,t}} c(P) = (c_1(P), \ldots, c_p(P))^\top = Cx, \tag{MOSP}$$

where $\mathcal{P}_{s,t}$ is the set of all paths from s to t in G, the matrix $C \in \mathbb{R}^{p \times n}$ is the cost matrix containing the rows c^k of coefficients of the p linear objective functions and $c_i(P) = \sum_{e \in P} c_e^i$ for $i \in \{1, \ldots, p\}$.

The single-objective variant ($p = 1$) can be solved in polynomial time using well-known algorithms such as Dijkstra's and Bellman-Ford algorithm. Dijkstra's algorithm, which is only suitable for graphs with non-negative arc weights, can be implemented to run in $O(m + n \log n)$ time using a priority queue (see, e.g., Schrijver [2003]). The Bellman-Ford algorithm, which handles graphs with negative arc weights, runs in $O(mn)$ time if no negative cycle exists. A *distance table* (or matrix) is a representation of the shortest distances between each pair of nodes and may be computed in time $\mathcal{O}(n^3)$ using the *Floyd-Warshall algorithm* or in time $\mathcal{O}\left(n^2 \log n + mn\right)$ by (essentially) repeated calls of Dijkstra's algorithm if the graph contains non-negative costs. The latter is more efficient on sparse graphs. A comprehensive overview of these Algorithms can be found in Cormen et al. [2001].

The shortest path problem can also be formulated as the following integer linear program:

$$\begin{aligned}
\min \quad & Cx \\
\text{s.t.} \quad & \sum_{a \in \delta^+(v)} x_a - \sum_{a \in \delta^-(v)} x_a = \begin{cases} 1 & \text{if } v = s \\ -1 & \text{if } v = t \\ 0 & \text{otherwise} \end{cases} && \forall\, v \in V \\
& x_e \in \{0, 1\} && \forall\, a \in A
\end{aligned}$$

where x_a is a binary variable indicating whether edge a is included in the path ($x_a = 1$) or not ($x_a = 0$). Any feasible solution x corresponds to an incidence vector of a path from s to t in G. The convex hull of the incidence vectors of all such paths is called the *path polytope*, which is an integer polyhedron [Schrijver, 2003]. Therefore, we can replace $x \in \{0, 1\}^m$ with $x_a \geq 0$ for all $a \in A$ and consider the linear relaxation problem to obtain an optimal solution.

The multi-objective minimum shortest path problem is known to be intractable. The finished decision problem and the canonical decision problem are known to be **NP**-hard, even in the case of two objectives.

Theorem 2.60 (Serafini, 1986)**.** *The finished bi-objective shortest path ($BOSP^{FIN}$) problem is* **NP***-complete.*

The **NP**-completeness proof is done by a polynomial time reduction from the Knapsack Problem to the BOSP$^{\text{FIN}}$ problem.

Theorem 2.61 (Hansen, 1980)**.** *The bi-objective shortest path problem is intractable.*

Proof. Consider a graph with an odd number of n nodes $V = \{v_1, \dots, v_n\}$ with $v_1 = s$ and $v_n = t$ and three arc types defined by

$$\begin{aligned} a &= (v_{2k-1}, v_{2k+1}), i = 1, 2, \dots, \tfrac{n-2}{2} && \text{with cost } c(a) = \left(2^{\frac{2k-2}{2}}, 0\right) \\ a &= (v_{2k-1}, v_{2k}), i = 1, 2, \dots, \tfrac{n-2}{2} && \text{with cost } c(a) = \left(0, 2^{\frac{2k-2}{2}}\right) \\ a &= (v_{2k}, v_{2k+1}), i = 1, 2, \dots, \tfrac{n-2}{2} && \text{with cost } c(a) = (0, 0). \end{aligned}$$

A visualization of this graph can be found in Figure 2.3. It holds that each path has costs

$$\sum_{a \in P} (c_1(a) + c_2(a)) = \sum_{i=1}^{\frac{n-2}{2}} 2^{\frac{2k-2}{2}} = \sum_{i=0}^{\frac{n-3}{2}} 2^i = 2^{\frac{n-1}{2}} - 1$$

and that for each $z \in \left\{0, \dots, 2^{\frac{n-1}{2}} - 1\right\}$ there is a path P from $v_1 = s$ to $v_n = t$ with $\sum_{a \in P} c_1(a) = z$ and $\sum_{a \in P} c_2(a) = 2^{\frac{n-1}{2}} - 1 - z$. Therefore all $2^{\frac{n-1}{2}}$ paths are efficient and $|\mathcal{Y}_N| = 2^{\frac{n-1}{2}}$. □

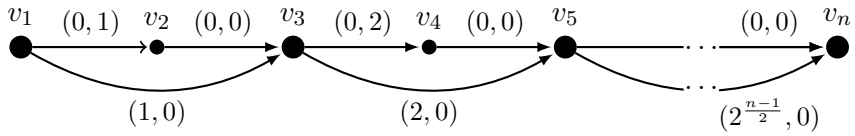


Figure 2.3: A graph with exponential many efficient solutions (paths). Note that the cost labels are transposed in this figure.

Theorem 2.62 (Bökler et al., 2017)**.** *There is no output-polynomial algorithm for the MOSP problem unless* $\mathbf{P} = \mathbf{NP}$*, even in the bi-objective case* ($p = 2$)*.*

Proof. For the proof, it is shown that the finished decision problem MOSP$^{\text{FIN}}$ is **co-NP**-hard. By Theorem 2.23, this indicates that an output-polynomial algorithm for MOSP implies $\mathbf{P} = \mathbf{co\text{-}NP}$, and therefore $\mathbf{P} = \mathbf{NP}$. This is shown by reducing instances of the complement of the Knapsack problem:

$$\left\{(w, p, k_1, k_2) : w^\top x \leq k_1, p^\top x \geq k_2, x \in \{0,1\}^n\right\}. \tag{KP}$$

Assuming without loss of generality that $w, p \in \mathbb{N}^n$ with $w_i, p_i > 0$ for all $i = 1, \dots, n$, $k_1, k_2 \in \mathbb{N}$, $w^\top \mathbf{1} > k_1$, and $p^\top \mathbf{1} > k_2$, the problem remains **NP**-complete under these conditions [Kellerer et al., 2004]. Consequently, the complement, **co**-KP, is **co-NP**-hard.

An instance $\hat{I}$ of the MOSP$^{\text{FIN}}$ problem is constructed from an instance I of the KP problem. The instance includes nodes $\{v_i^1, v_i^2\}$ for $i \in \{1, \ldots, n\}$, and an additional node v_{n+1}^1. For each item $i \in \{1, \ldots, n\}$, there is an arc $\left(v_i^1, v_i^2\right)$ with cost $(w_i, 0)^\top$, an arc $\left(v_i^1, v_{i+1}^1\right)$ with cost $(0, p_i)^\top$, and an arc $\left(v_i^2, v_{i+1}^1\right)$ 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 an additional arc (s, t) with cost $(k_1 + 1, 0)^\top$ and an additional path (s, v, t) with cost $\left(0, p^\top \mathbf{1} - k_2 + 1\right)^\top$. To complete the reduction, set $M \coloneqq \left\{(k_1 + 1, 0)^\top, \left(0, p^\top \mathbf{1} - k_2 + 1\right)^\top\right\}$. The construction of the graph is illustrated in Figure 2.4. The instance is valid, and there are at least two Pareto-optimal paths: (s, t) and (s, v, t), with costs $(k_1 + 1, 0)^\top$ and $\left(0, p^\top \mathbf{1} - k_2 + 1\right)^\top$, respectively. Both paths are Pareto-optimal because $w, p > 0$ for all $i \in \{1 \ldots, n\}$ and thus all other paths have either non-zero components in their objective function values or one objective equals the sum of all objective components and have value 0 in the other. All steps can be performed in polynomial time in the input instance I. Now, consider an instance $I \notin$ co-KP, or equivalently $I \in$ KP. There exists $x \in \{0, 1\}^n$ such that $w^\top x \leq k_1$ and $p^\top x \geq k_2 \Leftrightarrow p^\top (\mathbf{1} - x) \leq p^\top \mathbf{1} - k_2$. Using this solution, a path P in the MOSP instance $\hat{I}$ is constructed: For every i with $x_i = 1$, the route through node v_i^2 is taken, inducing a cost of $(w_i, 0)^\top$. For every i with $x_i = 0$, the direct route through arc $\left(v_i^1, v_{i+1}^1\right)$ is taken, inducing a cost of $(0, p_i)^\top$. It follows that $c_1(P) = w^\top x \leq k_1$ and $c_2(P) = p^\top (\mathbf{1} - x) \leq p^\top \mathbf{1} - k_2$. Hence, $\hat{I} \notin$ MOSP$^{\text{FIN}}$ since P is neither dominated by (s, t) nor (s, v, t). Suppose for some instance I, the constructed instance $\hat{I} \notin$ MOSP$^{\text{FIN}}$, meaning 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 be that $c_1(P) \leq k_1$ and $c_2(P) \leq p^\top \mathbf{1} - k_2$. A solution to KP in I can be constructed as follows: The path P cannot take arcs from (s, t) or (s, v, t), so it must take the route through the v_i^1 and v_i^2 nodes. For every $i \in \{1 \ldots, n\}$, it can either take arc $\left(v_i^1, v_i^2\right)$ or $\left(v_i^1, v_{i+1}^1\right)$. If it takes the first arc, set $x_i \coloneqq 1$; if it takes the second arc, set $x_i \coloneqq 0$. This solution then has cost $w^\top x = c_1(P) \leq k_1$ and $p^\top x = p^\top \mathbf{1} - c_2(P) \geq k_2$. Therefore, $I \in$ KP or equivalently $I \notin$ co-KP. This proves that the reduction is a polynomial-time reduction from the complement of KP to the finished decision variant of MOSP, thus establishing the theorem.

□

2.5 Single and Multi-objective Minimum Cost Flow Problem

This section focuses on the single and multi-objective minimum cost flow problems. For an in-depth overview of the single-objective minimum cost flow problem, the textbook by Ahuja et al. [1993] is recommended. Additionally, Hamacher et al. [2007b] comprehensively reviews the relevant literature on the multi-objective minimum cost flow problem.

The chapter begins by presenting the general notation and assumptions. Afterwards, the main properties and results are introduced, starting with the single-objective case and then extending to the multi-objective case.

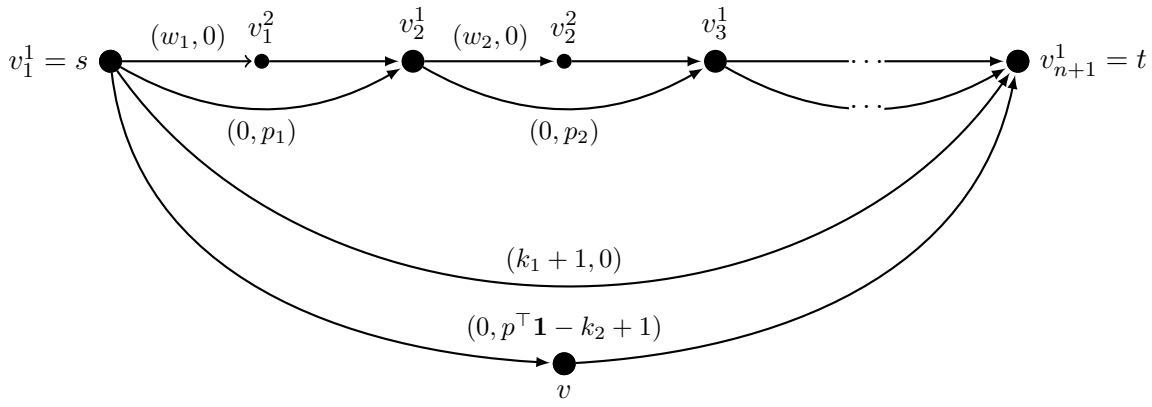


Figure 2.4: The reduction in the proof of Theorem 2.62 presented in Bökler et al. [2017]. Note that the cost labels are transposed in this figure.

2.5.1 General Notation and Assumptions

Flow A *network flow* problem involves moving a commodity through a network. Certain nodes supply the commodity, while other nodes demand it. The commodity is transported along the network's arcs, with the flow along these arcs constrained by specified lower and upper bounds, known as capacities.

Definition 2.63. *Let $D = (V, A)$ be a directed graph with n nodes $V = \{v_1, \dots, v_n\}$ and m arcs $A = \{e_1, \dots, e_m\}$. Let l_{ij} and u_{ij} denote the integer-valued, non-negative, finite lower and upper capacity bounds, respectively, for each arc $(i, j) \in A$. Here, b_i is the integer-valued* flow balance *of the node $i \in \{1, \dots, n\}$ where nodes with $b_i > 0$ are supply nodes, $b_i < 0$ are demand nodes, and $b_i = 0$ are transshipment nodes. A function $f\colon A \to \mathbb{R}_{\geq 0}$ with $f(i, j) = f_{ij}$ is called a feasible* flow *if f satisfies*

$$
\begin{aligned}
l_{ij} \leq f_{ij} \leq u_{ij} \quad & \text{for all } (i,j) \in A, && \text{(capacity constraint)} \\
\sum_{j:(i,j)\in A} f_{ij} - \sum_{j:(j,i)\in A} f_{ji} = b_i \quad & \text{for all } i \in V. && \text{(flow balance constraint)}
\end{aligned}
$$

If further $f = (f_{ij})_{ij \in A} \in \mathbb{Z}^m_{\geqq}$ the flow is called an integer flow.

Figure 2.5: Illustration of a node a and an arc $(1, 2)$, demonstrating the satisfaction of the flow balance and capacity constraints, respectively. In the figure on the left, the arcs are labeled with their corresponding flow values.

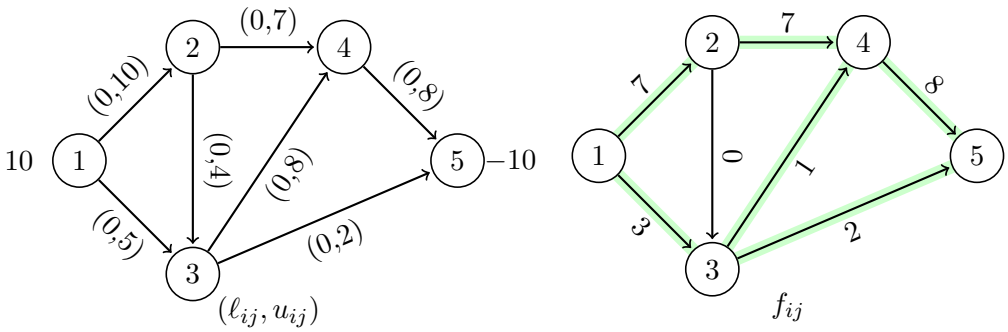


Figure 2.6: Example of a network with lower and upper bounds on the left. Nodes without labels are assumed to be transshipment nodes, i.e., $b_i = 0$. The goal is to send ten flow units from node 1 through node 5 across the network. The figure on the right shows a feasible flow in this network, with arcs labeled in green where the flow value is greater than zero.

Multi-Objective Minimum Cost Flow Problem Given a p-dimensional cost vector that assigns a cost vector to each arc, the multi-objective integer minimum cost flow problem asks for a feasible flow with minimal cost.

Definition 2.64. *Given a directed graph $D = (V, A)$, a cost function $c\colon A \to \mathbb{R}^p$ that assigns a cost vector to each arc, lower and upper capacity bounds, and a flow balance vector as above, the (multi-objective)* minimum integer cost flow *problem is defined as:*

$$\min_{f \in \mathcal{F}} c(f) = (c_1(f), \ldots, c_p(f))^\top = C \cdot f, \tag{MOIMCF}$$

where $\mathcal{F}$ is the set of all integer flows in D, $C \in \mathbb{R}^{p \times n}$ is the cost matrix containing the rows c^k of coefficients of the p linear objective functions and $c_i(f) = \sum_{a \in A} c_a^i f_a$ for $i \in \{1, \ldots, p\}$.

Note that the *continuous multi-objective minimum cost flow* (MOMCF) problem is the LP-relaxation of MOIMCF problem. Due to the total unimodularity of MOMCF, each extreme supported nondominated point of MOMCF has an integer preimage since u and b are integral (see Ahuja et al. [1993]). In other words, the sets of extreme supported nondominated points of MOMCF and MOIMCF, and thus the respective upper images, coincide. In the remainder of this thesis, only integer flows are considered, and from now on, flow always refers to an integer flow.

Assumptions In the context of the minimum cost flow problem, several common assumptions are made. It is assumed that the underlying directed graph D is connected. If the graph is not connected you can solve the problem on every connected component disjointly. Moreover, it is assumed that $\sum_{i \in V} b_i = 0$; otherwise the problem would be infeasible. It is further assumed that at least one feasible flow exists.

Without loss of generality, one can assume that l_{ij}=0 for all arcs $(i, j) \in A$. If this is not the case, a simple network transformation can be applied to achieve zero lower bounds.

For each arc with $l_{ij} > 0$, at least l_{ij} units of flow must be sent along this arc. To remove the lower bound, set the new lower bound $l'_{ij} = 0$ and reduce the capacity of the arc by the amount of the lower bound, i.e., $u'_{ij} = u_{ij} - l_{ij}$. Adjust the flow to account for the lower bound by introducing a new flow variable f'_{ij} for the transformed arc such that $f'_{ij} = f_{ij} - l_{ij}$, where f_{ij} is the original flow on arc (i, j). Next, update the demand at the nodes i and j to reflect the adjustments made to the arc, i.e. $b'_i = b_i - l_{ij}$ and $b'_j = b_j + l_{ij}$, see Ahuja et al. [1993].

It is assumed that every pair of nodes is reachable through a directed path in both directions. If necessary, this assumption can be enforced by adding artificial arcs (r, j) and (j, r) for each node $j \in V$, using an arbitrarily chosen but fixed root r and assigning a sufficiently high cost and capacity to each of these arcs. In any optimal solution, such arcs will only have flow greater than zero if the problem has no feasible solution without these arcs.

In the exposition of the thesis, it is assumed that D contains no parallel or anti-parallel arcs, i.e., if $(i, j) \in A$, then $(j, i) \notin A$, to avoid any confusion with the notation. However, the only difficulty with multiple arcs between a pair of nodes is keeping track of the relation that links residual arcs with original arcs, which can be done as explained in Sedeño-Noda and Espino-Martín [2013]. Modifying the algorithms in further sections is straightforward to deal with graphs containing multiple arcs between a pair of nodes.

2.5.2 Single-Objective Minimum Cost Flow Problem

The single-objective minimum cost flow problem is a fundamental and well-studied problem in combinatorial optimization. For the single-objective version, various polynomial and even strongly polynomial algorithms exist. For a comprehensive overview, see Ahuja et al. [1993] or Bertsekas [1998]. Before discussing the algorithms, the most important concepts and optimality criteria for network flows are stated. Many of these concepts can be transferred to the multi-objective case.

Residual graph The residual graph is central to many network flow algorithms, representing and dealing with the remaining capacity of arcs after accounting for the current flow.

Definition 2.65. *For a given network D, $D_f = (V, A_f := A^+ \cup A^-)$ is defined as the* residual graph *with respect to a feasible flow f, where*

$$A^+ := \{(i, j) \colon (i, j) \in A,\ f_{ij} < u_{ij}\} \quad \text{and } A^- := \{(j, i) \colon (i, j) \in A,\ f_{ij} > l_{ij}\}.$$

In D_f, the residual capacities *and* residual costs *are defined by $u_{ij}(f) := u_{ij} - f_{ij} > 0$ and $c_{ij}(f) := c_{ij}$, respectively, if $(i, j) \in A^+$ and $u_{ij}(f) := f_{ji} - l_{ji} > 0$ and $c_{ij}(f) := -c_{ji}$, respectively, if $(i, j) \in A^-$.*

The cost of a directed cycle C in D_f is given by $c(f,C) := \sum_{(i,j)\in C} c_{ij}(f)$ and let $\chi(C) \in \{-1,0,1\}^A$ be the (directed) incidence vector of C with

$$\chi_{ij}(C) := \begin{cases} 1, & \text{if } C \text{ traverses } (i,j) \text{ in } D_f, \\ -1, & \text{if } C \text{ traverses } (j,i) \text{ in } D_f, \\ 0, & \text{otherwise} \end{cases} \qquad \text{for all } (i,j) \in A.$$

We say that f' is obtained from f by *augmenting f along C by λ* if we set $f' := f + \lambda\chi(C)$. The cost $c(f')$ of f' is exactly $c(f) + \lambda c(f,C)$. An undirected cycle C_D in D is called *augmenting cycle* with respect to a flow f if whenever augmenting a positive amount of flow on the arcs in the cycle, the resulting flow remains feasible. Each augmenting cycle with respect to a flow f corresponds to a directed cycle C in D_f, and vice versa [Ahuja et al., 1993]. We call C_D the *equivalent* of C in D. Augmenting a flow f over a cycle $C \in D_f$ by one unit yields another feasible flow $f' = f + \chi(C)$. This is because, C is a directed cycle in D_f, augmenting along C increases and decreases flow on arcs in the cycle, maintaining flow conservation at each node. In addition, for any arc (i,j) in C, either (i,j) is in the original graph G and $f_{ij} < c_{ij}$ and hence $f_{ij} + 1 \leq c_{ij}$, or (j,i) is in G and $f_{ji} > 0$ and hence $f_{ji} - 1 \geq 0$. Thus, augmenting along C respects the capacity constraints.

The following fact is well-known for two feasible integer flows.

Property 2.66 (Schrijver, 2003)**.** *If f and f' are two feasible integer flows, then there are directed cycles $C_1, \ldots, C_k \subset D_f$ and integers $\lambda_1, \ldots \lambda_k > 0$ such that*

$$f' = f + \sum_{j=1}^{k} \lambda_j \chi(C_j),$$

$$c(f') = c(f) + \sum_{j=1}^{k} \lambda_j c(f, C_j).$$

Negative-Cycle Optimality Condition The first optimality condition can be stated using the residual graph. The *negative cycle optimality condition* states that a flow f is optimal if and only if D_f does not contain a negative cost cycle.

Theorem 2.67 (Ahuja et al., 1993)**.** *For a minimum cost flow problem with network D, a flow f is an optimal flow if and only if D_f does not contain a negative cost cycle.*

Proof. Let f be an optimal flow. If there exists a cycle C in D_f with $c(f,C) < 0$ exists, then the flow $f' = f + \chi(C)$ would be a feasible flow with $c(f') = c(f) + c(f,C) < c(f)$, which contradicts the optimality of f. Conversely, if D_f does not contain any negative cycle, then by Property 2.66, no feasible flow f' with less cost than $c(f)$ can exist. □

Duality For the following optimality conditions, concepts from duality theory will be used, see Section 2.2.2. To derive the dual, we associate dual variables π_i with the flow conservation constraints (one for each node $i \in V$) and τ_{ij} with the capacity constraints $f_{ij} \leq u_{ij}$ (one for each arc $(i,j) \in A$).

Definition 2.68. *The dual to the minimum cost flow problem is given by:*

$$\begin{aligned} \max \quad & \sum_{i \in V} \pi_i b_i - \sum_{(i,j) \in A} \tau_{ij} u_{ij} \\ s.t. \quad & \pi_i - \pi_j - \tau_{ij} \leq c_{ij} \qquad \forall (i,j) \in A \\ & \tau \geq 0. \end{aligned}$$

Note that the dual variables τ_{ij} in an optimal solution can always be chosen as $\tau_{ij} = \max\{0, -(c_{ij} + \pi_i - \pi_j)\}$ (see, Ahuja et al. [1993]). Hence, the dual problem reduces to finding a feasible vector π that minimizes the objective function $\sum_{i \in N} b_i \pi_i$.

Reduced Cost The dual variable π_i represents the *node potential* at node $i \in V$. For a given set of node potentials π the *reduced cost* $\bar{c}_{ij}$ of a given arc (i,j) are defined as

$$\bar{c}_{ij} := c_{ij} + \pi_i - \pi_j.$$

For forward arcs $(i,j) \in A^+$ of the residual network, the *residual reduced cost* is defined as $\bar{c}_{ij}(f) := c_{ij}(f) + \pi_i - \pi_j = c_{ij} + \pi_i - \pi_j$. For backward arcs $(j,i) \in A^-$, we define the residual reduced cost as $\bar{c}_{ji}(f) = c_{ji}(f) + \pi_j - \pi_i = -c_{ij} + \pi_j - \pi_i = -\bar{c}_{ij}$.

Property 2.69. *The cost of a cycle C in D_f is*

$$c(f,C) = \sum_{(i,j) \in C} c_{ij}(f) = \sum_{(i,j) \in C} c_{ij}(f) + \sum_{(i,j) \in C} (\pi_i - \pi_j) = \sum_{(i,j) \in C} \bar{c}_{ij}(f) = \bar{c}(f,C).$$

Therefore, a negative cycle C in D_f with negative costs $c(f,C)$ is also negative regarding the residual reduced costs $\bar{c}(f,C)$, implying an alternative form of the negative optimality condition, referred to as *reduced cost optimality condition.*

Theorem 2.70 (Ahuja et al., 1993)**.** *A feasible flow f is optimal for the minimum cost flow problem if and only if corresponding node potentials π exist that satisfy the following reduced cost optimality condition:*

$$\bar{c}_{ij}(f) \geq 0 \quad \forall (i,j) \in A_f.$$

Complementary Slackness Condition While the negative cycle and reduced cost optimality conditions apply to the residual graph, the *complementary slackness optimality condition* reformulates these conditions in terms of the original graph, based on the complementary slackness optimality conditions of linear programming.

Theorem 2.71 (Ahuja et al., 1993)**.** *A feasible flow f is an optimal flow of the minimum cost flow problem if and only if feasible node potentials π exist such that the reduced costs and flow values satisfy the complementary slackness optimality conditions for every arc $(i,j) \in A$:*

$$\begin{aligned} &(i) \quad \text{If } \bar{c}_{ij} > 0, \text{ then } f_{ij} = 0. \\ &(ii) \quad \text{If } 0 < f_{ij} < u_{ij}, \text{ then } \bar{c}_{ij} = 0 \\ &(iii) \quad \text{If } \bar{c}_{ij} < 0, \text{ then } f_{ij} = u_{ij}. \end{aligned}$$

Any node potential π that satisfies the complementary slackness optimality conditions, and consequently the reduced cost optimality conditions, is *optimal*. For a given optimal flow f, an optimal node potential π can be determined by computing the shortest path distances in D_f from an arbitrary but fixed root node r to all other nodes $i \in V$. Since it is assumed that every pair of nodes is reachable through a directed path in both directions and given that f is optimal (implying no negative cycles exist in D_f), all nodes $i \in V$ are reachable from r in D_f. Therefore, the shortest path distances are well-defined.

Tree Solution The concepts of *tree solutions* and *induced cycles* are central to numerous algorithms, including the well-known *network simplex algorithm* first presented in Dantzig [1951], and will be important for the algorithms discussed in the following chapters. Any basic feasible solution f of MCF corresponds to a (not necessarily unique) spanning-tree structure (T, L, U). For simplicity, we shall abbreviate *spanning tree* as *tree* in the sequel and refer to a tree T when we mean its set of arcs.

Definition 2.72. *The flow f and an associated* tree structure (T, L, U) *is called a* tree solution *if it consists of a spanning tree T of D and a disjoint set $A \setminus T = L \dot\cup U$ such that the flow f satisfies*

$$f_{ij} = l_{ij} \quad \text{for all } (i,j) \in L,$$
$$f_{ij} = u_{ij} \quad \text{for all } (i,j) \in U.$$

If a feasible (optimal) flow exists, then there also exists a feasible (optimal) tree solution [Cook et al., 1998], respectively. The network simplex algorithm always determines an optimal tree solution [Ahuja et al., 1993].

The negative optimality condition states that a flow f is optimal if and only if D_f does not contain a negative cycle and a tree solution helps us find cycles in D_f quickly: Indeed, given a tree solution f with an associated tree structure (T, L, U), it suffices to check cycles that arise by inserting a single edge of $A \setminus T$ into T in order to prove the optimality of f, see Cook et al. [1998] for details. This verification can be further simplified if a node potential is considered, i.e., a vector $\pi \in \mathbb{R}^V$, where π_v is defined to be the cost of the (unique) simple undirected path in T from an arbitrary but fixed node r to v. Here, the cost of an arc $(i,j) \in D$ traversed in forward direction by the path contributes positively to the total cost, whereas the cost of a backwards-traversed arc contributes negatively. Then, the following property holds:

Property 2.73 (Ahuja et al., 1993)**.** *A tree solution f with an associated tree structure (T, L, U) is optimal if*

$$(i) \quad \bar{c}_{ij} = 0 \quad \text{for all } (i,j) \in T,$$
$$(ii) \quad \bar{c}_{ij} \geq 0 \quad \text{for all } (i,j) \in L,$$
$$(iii) \quad \bar{c}_{ij} \leq 0 \quad \text{for all } (i,j) \in U.$$

These three conditions are equivalent to $\bar{c}_{ij}(f) \geq 0$ for all $(i,j) \in D_f$. The equation (i) is always true by the definition of the node potential π. With conditions (ii) and (iii), it follows that any cycle in D_f has a non-negative cost. Hence, f is an optimal flow. The overall advantage of this kind of node potential is that we can construct cycles by inserting one single edge to T with some nice properties, which we will introduce below.

Definition 2.74 (Cook et al., 1998)**.** *Let $(i,j) \notin T$ be a non-tree arc.*

(i) There exists a unique *cycle $C(T,L,U,(i,j))$* induced by *(i,j), that is formed by (i,j) together with the j-i path in T, designated with $P^T_{j,i}$. This cycle is from now on referred to as C_{ij}.*

(ii) The arc (i,j) defines the orientation of the cycle C_{ij}. If $(i,j) \in L$, then the orientation of the cycle is in the same direction as (i,j). If otherwise $(i,j) \in U$, then the cycle's orientation is in the opposite direction. We define the set of all arcs (u,v) in C_{ij} that are in the same direction as the orientation of the cycle with $\overrightarrow{C_{ij}}$ and the set of all arcs (u,v) that are opposite directed with $\overleftarrow{C_{ij}}$.

See Figure 2.7 for an illustration.

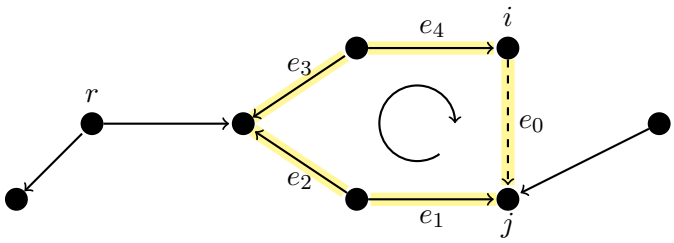


Figure 2.7: Example of the unique cycle C_{e_0} induced by e_0. Non-tree arcs are drawn as dashed lines. In this example $e_0 \in L$ and therefore $\overrightarrow{C_{e_0}} = \{e_0, e_2, e_4\}$ and $\overleftarrow{C_{e_0}} = \{e_1, e_3\}$.

Property 2.75 (Cook et al., 1998)**.** *Given a tree structure (T,L,U), the unique cycle C_{ij} induced by an arc $(i,j) \notin T$ satisfies:*

- *For all arcs $(u,v) \in C_{ij}$ it holds that $(u,v) \in T \cup (i,j)$.*
- *The cost of C_{ij} is given by $c(C_{ij}) = \bar{c}_{ij}$ if $(i,j) \in L$ and $c(C_{ij}) = -\bar{c}_{ij}$ if $(i,j) \in U$.*

Solution Methods Minimum cost flow problems can be solved using a variety of polynomial time and strongly polynomial time algorithms, each designed to efficiently identify the optimal flow. These methods include capacity scaling, cost scaling, simplex methods, cycle-canceling techniques, relaxation algorithms, out-of-kilter algorithms, and primal-dual approaches. Below is a brief overview of some key algorithms, with a more comprehensive discussion of nearly all solution techniques available in Ahuja et al. [1993].

1. *Cycle-Canceling Algorithm:* Introduced by Klein [1967], this algorithm is one of the earliest methods for solving the minimum cost flow problem by systematically eliminating negative cost cycles in the residual network. The algorithm incrementally reduces the overall flow cost by iteratively identifying and canceling these cycles. An extension, known as the mean cycle-canceling algorithm, was presented in Goldberg and Tarjan [1989], which focuses on identifying and canceling cycles with the

minimum mean cost in the residual network. This extension results in a strongly polynomial time algorithm with a running time of $\mathcal{O}(n^2m^3 \log n)$, see Ahuja et al. [1993].

2. *Network Simplex Method:* An adaptation of the simplex method for network flow problems, first presented in Dantzig [1951]. Later, several authors improved the algorithm. In Orlin [1997], the first polynomial runtime algorithm with a complexity of $\mathcal{O}(n^2m \log(n\zeta))$, where ζ represents the maximum cost of any edge was presented. This was further refined to $\mathcal{O}(nm \log n \log(n\zeta))$ in Tarjan [1997]. It operates on feasible spanning tree structures of the network, pivoting between these trees to find the minimum cost flow. It makes use of the induced cycles of a spanning tree, helping find a negative cycle in the graph by just inserting a single edge of a non-tree arc into the tree. The algorithm always determines an optimal tree solution and has proven to be highly efficient in practice.

3. *Enhanced Capacity Scaling Algorithm:* Developed by Orlin [1993], this algorithm builds on earlier capacity scaling methods by incorporating advanced scaling techniques that enhance efficiency. It reduces the number of augmenting path computations, solving the minimum cost flow problem as a sequence of $\mathcal{O}(m \log n)$ shortest path problems. This approach achieves a worst-case running time of $O\big((m \log n)(m + n \log n)\big)$, making it a strongly polynomial time algorithm [Ahuja et al., 1993].

2.5.3 Multi-Objective Minimum Cost Flow Problems

As discussed, various polynomial algorithms exist for the single-objective version of the minimum integer cost flow problem. However, real-world scenarios often involve multiple objectives, which may be conflicting, making it impossible to find a single solution that optimizes all objectives simultaneously. This typically results in many incomparable solutions. These *multi-objective integer minimum cost flow* (MOIMCF) problems are significantly more challenging to solve, as reviewed in Hamacher et al. [2007b], where the authors highlight a lack of algorithms designed explicitly for MOIMCF.

The MOIMCF problem can be relaxed to a linear program by removing the integrality constraints on flow values. This LP relaxation is denoted as the *multi-objective minimum cost flow* (MOMCF) problem. With the relaxed problem, only *supported nondominated points* of MOIMCF can be obtained. However, in the original MOIMCF problem, both *nonsupported* and *supported nondominated points* can be present. Figure 2.8 presents a bi-objective example along with the corresponding solution types. Figure 2.9 illustrates the outcome space for the example given in Figure 2.8.

Hardness and Intractability The multi-objective shortest path problem (MOSP) is a special case of the multi-objective integer minimum cost flow problem [Ahuja et al., 1993]. The s-t path problem can be transformed into a minimum cost flow problem by setting all arc capacities to one and sending one unit of flow from s to t, with $b_s = 1$, $b_t = -1$, and $b_v = 0$ for all $v \in V \backslash \{s, t\}$. In this modified network, an optimal flow corresponds to the shortest path from s to t. Consequently, the finished decision problem and canonical

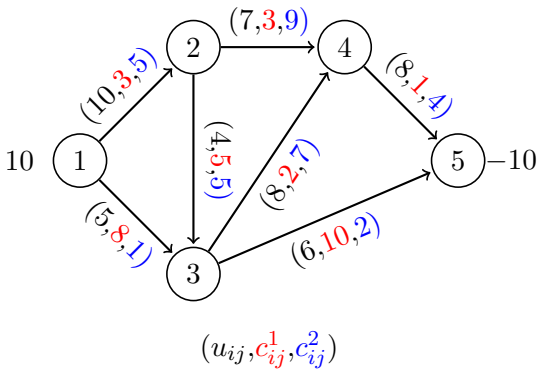


Figure 2.8: Illustration a bi-objective integer minimum cost flow problem. The example includes ten distinct nondominated points, and 83 non-efficient flows. The output space is visualized in Figure 2.9.

decision problem of the multi-objective integer minimum cost flow problem are **NP**-hard and the enumeration problem intractable even in the case of two objectives. The proofs follow directly from Theorem 2.60 and Theorem 2.61, respectively.

Theorem 2.76. *The finished bi-objective minimum cost flow problem is* **NP***-complete.*

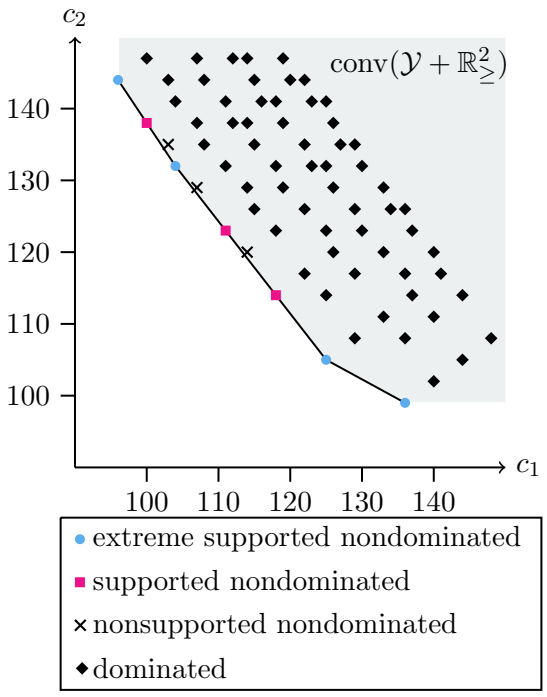
Theorem 2.77. *The bi-objective minimum integer cost flow problem is intractable.*

The number of supported nondominated points, even note that the number of extreme supported nondominated points, of a bi-objective integer minimum cost flow problem can grow exponentially in the input size. Thus, the enumeration problems of determining all supported nondominated points or extreme supported nondominated points are intractable. In the example provided by Hansen [1980] for Theorem 2.61, the objective function values for an exponential number of efficient paths are located on a single line. Specifically, all $2^{\frac{n-1}{2}}$ nondominated points in this instance are supported, leading to an exponential number of supported solutions in the objective space. Furthermore, Ruhe [1988] proves the existence of bi-objective integer minimum cost flow problem instances with 2^n extreme supported nondominated points. This establishes the following theorem:

Theorem 2.78 (Ruhe, 1988)**.** *The number of (extreme) supported nondominated points in a bi-objective integer minimum cost flow problem can be exponential in the input size.*

Solution Methods Hamacher et al. [2007b] provide an overview of the literature of exact and approximation methods for solving MOIMCF and MOMCF problems. Approximation algorithms typically compute a subset of the efficient solutions. The authors also observe that the literature generally focuses on bi-objective problems and highlights a gap in the availability of algorithms on MOIMCF problems with more than two objectives.

Any approach for MOILP or MOCO problems, as discussed in Section 2.3, can be applied to solve MOIMCF problems, including generic scalarization-based methods, two-phase approaches, and the dual variant of the Benson algorithm. The latter can be used to



c_1	c_2	Type
96	144	extreme
100	138	supported
103	135	nonsupported
104	132	extreme
107	129	nonsupported
111	123	supported
114	120	nonsupported
118	114	supported
125	105	extreme
136	99	extreme

Figure 2.9: Illustration of an upper image $\mathcal{Y}^{\geqq} = \mathrm{conv}(\mathcal{Y} + \mathbb{R}^2_{\geq})$ and the different solution types of the given BOIMCF problem in Figure 2.8. Note that the origin of this plot is at $(90, 90)^\top$ and not all dominated vectors are displayed in the figure.

determine the set of extreme supported nondominated points. However, a few specialized algorithms exploit the network structure of BOIMCF problems, and even fewer for the multi-objective variant of the problem. Some of them are presented below.

For bi-objective minimum cost flow problems, the parametric network flow algorithm can be effectively used in the first phase of two-phase methods. For instance, Raith and Ehrgott [2009] apply the Two-Phase Method to solve BOIMCF, utilizing a parametric network simplex method in the first phase to identify extreme supported solutions. This is followed by a flow ranking algorithm in the second phase, such as the one proposed by Hamacher [1995], (or the improved version that will be presented in Chapter 6), which can be used to generate the remaining supported and nonsupported solutions.

Eusébio and Figueira [2009a] propose an algorithm for BOIMCF that integrates the ε-constraint method with a branch-and-bound approach. In the multi-objective case, Eusébio and Figueira [2009b] compute the complete set of supported efficient integer solutions for MOIMCF problems. They demonstrate that these supported solutions are connected by chains of zero-cost cycles in the residual graphs.

Other methods, such as those presented in Sun [2011] and Eusébio et al. [2014], offer approaches for identifying subsets or representations of MOIMCF problems. However, a detailed discussion of these methods is beyond the scope of this thesis, as our focus lies

on the output-sensitivity analysis of different solution concepts.

Output-Sensitivity Analysis This thesis focuses on the output-sensitive analysis for different solution concepts. Let C be the configuration set of all supported efficient solutions and C^* the configuration set of all nondominated points. As mentioned before, an output-polynomial time algorithm $E = (I, C)$, which determines all supported efficient solutions exactly once, would also yield all nondominated points. However, it might not be output-polynomial w.r.t. this task, as there might be an exponential number of efficient solutions mapping to a small number of nondominated points. Thus, the algorithm for $E^* = (I, C^*)$ may output the elements (i.e., the nondominated points) more than once or even exponentially many times. Consider, for example, a directed graph with $\{1, \dots, n\}$ transshipment nodes, a node s and t with flow balance n and $-n$, respectively. The graph contains the arcs (s, i) and (i, t) for all $i \in \{1, \dots, n\}$ with upper capacity n. The cost of all arcs is equal to $(1, 1)^\top \in \mathbb{R}^2$. Then we have $\binom{2n-1}{n}$ supported efficient solutions, but all map to the same extreme supported nondominated point. Figure 2.10 illustrates this construction with $n = 4$ nodes.

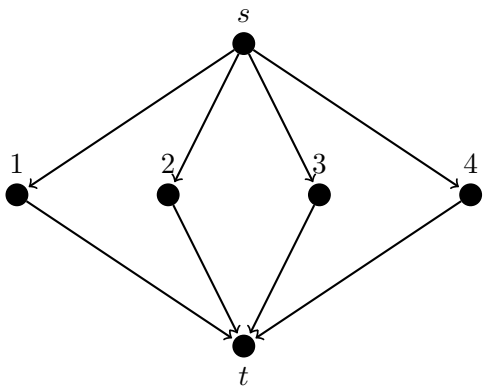


Figure 2.10: Example of a BOIMCF with $n = 4$ transshipment nodes, one supply node s with $b_s = 4$, and a demand node t with demand $b_t = -4$. All arcs have a capacity of $n = 4$ and costs equal to $(1, 1)^\top$. The problem contains $\binom{2n-1}{n} = \binom{7}{4} = 35$ supported efficient solutions, all mapping to a single nondominated point.

3 Generic Scalarization Based Algorithms

This chapter presents scalarization-based algorithms for multi-objective combinatorial optimization. It focuses on methods that decompose the overall problem into a series of scalarized single-objective subproblems, which can then be solved using available single-objective (IP-)solvers. The presented methods are generic, i.e., they are independent of specific problem structures and hence generally applicable. They exhibit flexibility by accommodating different implementation possibilities, including choices of the sequence of subproblems, scalarization strategies, and decisions regarding the computation of the complete or a representation of the nondominated point set.

Furthermore, the chapter references foundational results and algorithms, establishing connections with various aspects such as search region, search zones, local upper bounds, defining points, complexity results, and redundancy avoidance. It emphasizes the importance of these generic approaches and underscores their efficiency in solving multi-objective combinatorial optimization problems while exploring a wide range of options for their implementation.

To keep the notation in this chapter simple, we will not always distinguish between row and column vectors in the following. In particular, when listing outcome vectors for specific problem instances, we will often write these vectors as row vectors and omit the transposed sign whenever there is no risk of misinterpretation.

This chapter is based on a book chapter, co-authored with Kathrin Klamroth and Kerstin Dächert, which appears in Dächert et al. [forthcoming].

3.1 Introduction

Combinatorial optimization (CO) problems (not necessarily with multiple objectives) have a specific structure, wherefore they are often solved by very specific solution methods. Prominent examples are the Hungarian method for assignment problems, Dijkstra's algorithm for shortest path problems, Prim's and Kruskal's algorithms for minimum spanning tree problems, and the Chinese Postman method for shortest closed paths or circuits. However, it is also possible to apply rather generic methods that are applicable to general discrete (integer) optimization problems, such as the well-known branch-and-bound or branch-and-cut methods. Efficient implementations of such methods are readily available in solvers such as CPLEX and Gurobi.

MOO problems (not necessarily combinatorial) can be approached in various ways. A generic approach is a so-called scalarization methods that transform the given multi-objective optimization problem into one – or a series of – associated single-objective problems with modified (i.e., scalarized) objective functions and, potentially, additional constraints and variables. Scalarizations usually support the selection of parameters that

somehow control how the original objectives are embedded into the transformed problem. There are ways to combine all objectives into one new objective. There are also scalarization techniques in which some or all original objectives appear as new constraints. This chapter discusses the corresponding formulations as well as theoretical properties in detail.

On this basis, we approach MOCO problems from a quite general perspective. However, the combinatorial structure of MOCO problems yields discrete nondominated sets, a property that is heavily exploited in this chapter.

In particular, this chapter describes generic scalarization based algorithms for MOCO problems that combine scalarization techniques with generic single-objective solvers. Applying a generic single-objective solver is appealing since highly efficient implementations are available and they are independent of specific problem structures. Indeed, the transformation inherent in standard scalarization methods typically destroys or alters the combinatorial structure of the problem.

A few considerations should be made that lead to different recommendations concerning the choice of a specific scalarization technique. First, it is important to determine whether the goal is to generate the entire nondominated set or only a subset (i.e., a representation) of it. Second, one should decide whether the scalarization needs to be complete, i.e., whether every nondominated point needs to be determinable with an appropriate choice of the scalarization parameters. The latter usually requires more complex scalarizations and, hence, in general, longer solution times. Some methods switch from "easy" scalarization (in the early phases of an optimization process) to more "complex" scalarizations later (to identify potentially missing nondominated points). The two-phase method is a prominent example of this approach. Even if the two questions formulated above seem to be closely related, they should be answered separately.

As often the case in MOO, problems with two objectives are structurally more straightforward than those with three or more objectives. Indeed, the nondominated set of bi-objective problems can always be sorted such that the objective values of the first objective are increasing while those of the second objective are decreasing. This significantly simplifies any decomposition strategy in the objective space. In this chapter, we focus on MOO problems in arbitrary dimensions. Nevertheless, the bi-objective case remains important for visualization purposes and for explaining complex relations, at least partially. Therefore, this chapter also presents several examples and figures with two objectives in this chapter, keeping in mind that this is the "easy" case. While linking the two solution concepts for CO and MOO is relatively straightforward in the bi-objective case, it is not evident how to design such an algorithm for three or more objectives in an efficient way, that is, to solve as few optimization problems as possible, to avoid the regeneration of already known nondominated points, to keep the storage for computation small, and so on.

The remainder of this chapter is structured as follows. Sections Section 3.2 and Section 3.3 present the preliminaries for this chapter. In Section 3.2, two scalarization techniques well-suited for a generic MOCO approach are introduced and discussed. Section 3.3 then introduces the concepts of *search regions* and *search zones*. A comprehensive overview

of the literature on generic scalarization-based algorithms is provided in Section 3.4. Finally, Section 3.5 presents generic scalarization approaches for bi-objective and multi-objective combinatorial optimization problems, along with examples of these methods.

3.2 Scalarization Methods

The conventional strategy for addressing MOO problems involves scalarization. These methods convert multi-objective optimization problems into corresponding single-objective optimization problems, thereby facilitating their solution using standard optimization solvers.

Ideally, scalarization methods should exhibit the following desirable properties (see, e.g., the comprehensive surveys of Ehrgott [2006] and Miettinen and Mäkelä [2002]):

Correctness: An optimal solution of the scalarization is always (weakly) efficient.

Completeness: Every non-dominated point can be identified through an appropriate selection of scalarization parameters.

Simplicity: The scalarized problem is not more complex to solve than the corresponding single-objective problem.

Linearity: If the constraints and objectives of a MOCO problem are linear, the scalarized counterparts should also maintain linearity.

Depending on the application context (and on the MOCO problem at hand), this list may be extended by further properties.

Various scalarization techniques exist, each possessing its unique set of desirable properties. In addition to the weighted-sum scalarization presented in Definition 2.43, this section presents and discusses two scalarizations in more detail that are particularly well suited in the context of a generic scalarization based algorithm.

3.2.1 Epsilon-Constraint Scalarization

The Epsilon-constraint (also ε-constraint or budget-constraint, see, Haimes et al. [1971]) scalarization consists in selecting one objective function z_j, where $j \in 1, \ldots, p$, for minimization while imposing upper bounds $\varepsilon_i \in \mathbb{R}$ on all other objective functions z_i, with $i \in \mathcal{I}_j := \{1, \ldots, p\} \setminus \{j\}$:

$$\min_{x \in \mathcal{X}} \{z_j(x) : z_i(x) \leq \varepsilon_i \text{ for } i \in \mathcal{I}_j\} \tag{3.1}$$

The following properties are well-known [Ehrgott, 2005]:

Lemma 3.1. *Every optimal solution $\bar{x} \in X$ of an ε-constraint scalarization* (3.1) *is weakly efficient for MOCO, and at least one of the optimal solutions of* (3.1) *is efficient for MOCO.*

Weakly efficient solutions can be avoided by adopting a two-stage procedure in the form of:

$$\min_{x\in\mathcal{X}} \left\{ \sum_{i=1}^{p} z_i(x) : z_j(x) = z_j(\bar{x}), z_i(x) \leq \varepsilon_i \text{ for } i \in \mathcal{I}_j \right\} \tag{3.2}$$

over the set of optimal solutions of Problem (3.1).

Note that only optimal solutions of (3.1) are feasible for (3.2). However, this two-stage approach, in general, requires two solver calls. This can be avoided by alternatively adding an appropriate augmentation term to Problem (3.1) in the first stage, leading to an augmented ε-constraint scalarization

$$\min_{x\in\mathcal{X}} \left\{ z_j(x) + \rho \sum_{i=1}^{p} z_i(x) : z_i(x) \leq \varepsilon_i \text{ for } i \in \mathcal{I}_j \right\} \tag{3.3}$$

which returns an efficient solution of MOCO. Here, $\rho > 0$ is an appropriately chosen small augmentation parameter; see, e.g., Ehrgott and Ruzika [2008] and Mavrotas [2009]. Hence, ε-constraint scalarization ensures completeness, correctness, and linearity, rendering it one of the most commonly employed scalarization methods.

However, a drawback of the ε-constraint scalarization is that the additional budget constraints often increase the problem's computational complexity. For MOCO problems, the ε-constrained scalarizations are often **NP**-hard, even if the corresponding single-objective problems are polynomial solvable (e.g., in minimum-cost-flow problems, shortest-path-problems, etc.), as discussed in Ehrgott [2006]. Therefore, the method lacks the simplicity property.

3.2.2 Weighted Distance and Reference Point Methods

In the realm of *compromise programming methods*, the goal is to seek solutions as close as possible to a pre-specified *reference point* $z^R \in \mathcal{Y}$. While the Ideal point z^I would be the best possible outcome, yet impossible to obtain, this point or an *utopia point* that slightly dominates the Ideal point in all components is often chosen as a reference point. Given a distance measure $d : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}_{\geqq}$ the compromise programming problem can be formulated as:

$$\min_{x\in\mathcal{X}} d\left(z(x), z^I\right). \tag{3.4}$$

Commonly employed distances are from the class of ℓ_q-norms (typically called ℓ_p-norms in the mathematical context; however, p already denotes the number of objectives here). Alongside the reference point z^*, and a weight vector $\lambda \in \mathbb{R}^p_{>}$ reflecting the relative importance of the individual objective functions, the *weighted l_q-norm scalarization* with $1 \leq q < \infty$ can be expressed as:

$$\min_{x\in\mathcal{X}} \left(\sum_{i=1}^{p} \lambda_i |z_i(x) - z_i^R|^q \right)^{\frac{1}{q}}. \tag{3.5}$$

Special cases, particularly q = 1 and $q \to \infty$, hold significant importance. For $q \to \infty$, a notable special case emerges as the *weighted Tchebycheff scalarization* given by:

$$\min_{x \in \mathcal{X}} \max_{i \in \{1,\dots,p\}} \lambda_i |z_i(x) - z_i^R|. \tag{3.6}$$

When z^R aligns with the Ideal point (or a point dominating the Ideal point), the absolute values in equations (3.5) and (3.6) can be omitted. This simplification renders problem (3.5) equivalent to a weighted sum scalarization for $p = 1$, and problem (3.6) can be reformulated as:

$$\min_{x \in \mathcal{X}} \{\alpha : \lambda_i (z_i(x) - z_i^R) \leq \alpha \quad \forall i \in \{1, \dots, p\}\} \tag{3.7}$$

We refer again to Ehrgott [2005] for the following result.

Lemma 3.2. *If z^R is the ideal point or a utopia point, then every optimal solution $\bar{x} \in X$ of a weighted Tchebycheff scalarization* (3.6) *is weakly efficient for MOCO, and at least one of the optimal solutions of* (3.6) *is efficient for MOCO.*

Similar to the ε-constraint scalarization, weakly efficient solutions can be mitigated by either solving a second-stage problem or incorporating an augmentation term. However, selecting an appropriate augmentation parameter is crucial to avoid numerical issues or missing efficient solutions, as detailed in Dächert et al. [2012].

If the reference point z^R is the ideal point or a utopia point, then the weighted Tchebycheff scalarization (3.6) is correct, complete, and linear. However, as in the case for the epsilon-constraint scalarization, weighted Tchebycheff scalarizations are in general not simple due to the introduction of additional constraints.

Figure 3.1 illustrates the additional constraints added to the problem in the respective scalarization. In addition, exemplary level curves of the respective objective functions are shown.

Remark We acknowledge the existence of numerous other scalarization approaches, both related and unrelated, each with its unique set of advantages and limitations. Among them are methods such as *Benson's method* [Benson, 1978], *the direction method of Pascoletti and Serafini* [Pascoletti and Serafini, 1984], and *hypervolume scalarization* [Paquete et al., 2022]. Every scalarization method that is correct and complete can be used within a generic MOCO approach.

3.3 Search Region and Search Zones

In the previous section, we discussed two correct and complete scalarizations methods, i.e., scalarizations that enerate one (weakly) non-dominated point for each set of parameters (or show that none exists wrt. the given parameter settings). However, in general, not only one nondominated point but a (or the) set of nondominated points shall be generated. If the points are computed in a sequence and not at once (e.g., in parallel implementations),

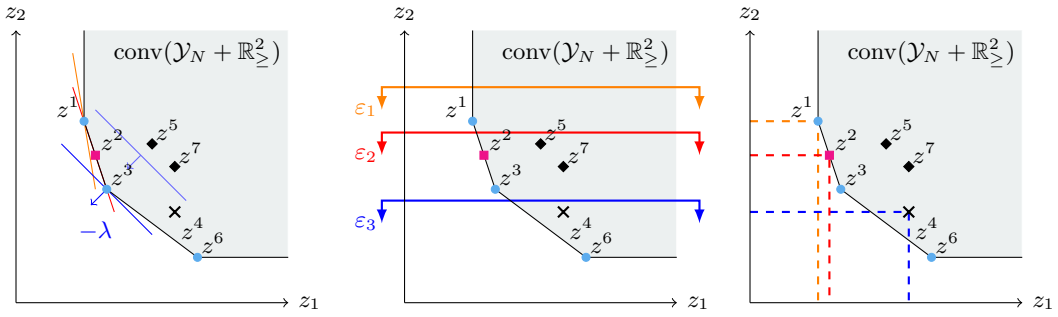


Figure 3.1: The weighted-sum method is illustrated on the left, with three different weights represented by distinct colors. The (ε-constraint method) is shown in the middle, illustrated with three different ε-constraints. Extreme supported nondominated points are marked with filled cycles, supported nondominated points with unfilled cycles, nonsupported non-dominated points with squares, and dominated points with crosses. Additionally, the (weighted Tchebycheff method) is depicted with various weights and level curves passing through z^1, z^2, and z^4. Using the ε-constraint method with these constraints, the points z^1, z^2, and z^4 are also images of the produced solutions, respectively. However, in the weighted-sum method, a single-objective problem with weights corresponding to the orange line would yield the point z^1. For the red line, the single-objective problem would produce z^1, z^2, and z^3 as images of optimal solutions. Similarly, a solution with z^4 as the image would be optimal for a single-objective problem with weights corresponding to the blue line. Importantly, no weights in the weighted-sum scalarization would make z^4 an image of any optimal solution.

we can use information from the already generated points. More precisely, we can exclude the part of the outcome space that is dominated by an already generated point. We call the remaining space *search region* since our search for further nondominated points concentrates on this area. The search region can be expressed as the union of hypercubes or hyperrectangles, called *search zones* in the following. Each search zone can be described by an upper bound vector, which we call *local upper bound* in the context of the search region. Below, we provide formal definitions of these concepts as in Dächert et al. [2024].

Definition 3.3. *For a given set of outcome vectors $N \subseteq \mathcal{Y}$, the search region $S(N)$ describes the part of $\mathbb{R}^p$ that is* not *dominated by any point of N, i.e., the part that potentially contains* further *non-dominated points. Formally, it is defined as*

$$\begin{aligned} S(N) &= \mathbb{R}^p \setminus \left(N + \mathbb{R}^p_{\leqq} \right) \\ &= \mathbb{R}^p \setminus \{ z \in \mathbb{R}^p : \exists\, \bar{z} \in N \text{ with } \bar{z} \leqq z \}. \end{aligned}$$

A concise characterization of $S(N)$ can be achieved by using a finite set of axis-parallel hyperrectancles called *search zones*, induced by a finite set of *local upper bounds* $U(N)$.

A local upper bound $u \in U(N)$ is a maximal point in the objective space that is not dominated by any point of N. Following the exposition in Klamroth et al. [2015], we assume that the initial search region is restricted to a bounding box with lower bound $(m, \ldots, m)^T \in \mathbb{Z}^p$ and an upper bound $(M, \ldots, M)^T \in \mathbb{Z}^p$ (with $m < M$). These bounds may be derived, for example, from global lower and upper bounds on the attainable objective function values or provided by the decision maker. We include p *dummy points* $d^i = Me^i \in \mathbb{Z}^p$ for $i = 1, \ldots, p$ in the set N, i.e. $d^i \in N$, to delimit the bounding box, where e^i denotes the i-th unit vector in $\mathbb{R}^p$.

Definition 3.4. *Every local upper bound $u \in U(N)$ defines a* search zone

$$C(u) = \{z \in \mathbb{R}^p : z < u\} = u - \mathbb{R}^p_>$$

Thus, the search region $S(N)$ can be described by the union of the *search zones* induced by a finite set of *local upper bouds* $U(N)$ such that

$$\begin{aligned} S(N) &= \{z \in \mathbb{R}^p : \exists u \in U(N), z < u\} \\ &= \bigcup_{u \in U(N)} C(u). \end{aligned}$$

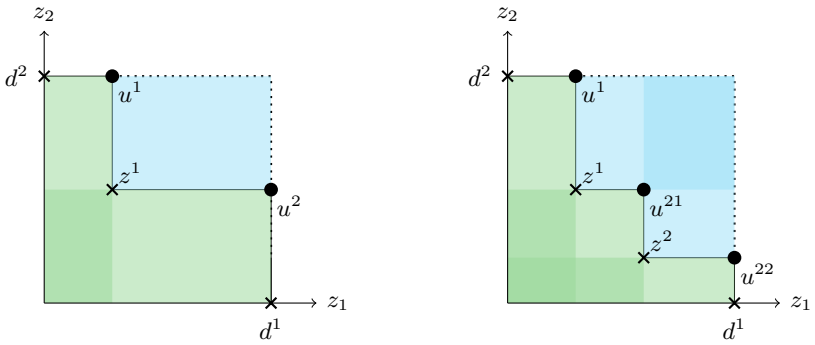


Figure 3.2: Dominated area in (displayed in blue), search zones (displayed in green), and local upper bounds in the bi-objective instance after adding the first (left figure) and the second non-dominated point (right figure).

The set $U(N)$ requires updating whenever a new non-dominated point $\bar{z}$ is found. Efficient operations to iteratively update the set $U(N)$ and detect redundant local upper bounds when further non-dominated points are added to N have been described by Klamroth et al. [2015] and Dächert et al. [2017], and will be discussed in Section 3.5.

Figure 3.2 and Figure 3.3 display the search region and the upper bound set with respect to one or two non-dominated points for example instances with two and three objective functions, respectively.

Example 3.5 (Klamroth et al., 2015)**.** *To illustrate the concept of local upper bounds consider the set of the two non-dominated points $z^1 = (3, 5, 7)^\top$ and $z^2 = (6, 2, 4)^\top$ yielding the local upper bounds $u^1 = (3, M, M)^\top, u^{21} = (6, 5, 10)^\top, u^{22} = (10, 2, 10)^\top, u^{31} = (6, 10, 7)^\top,$*

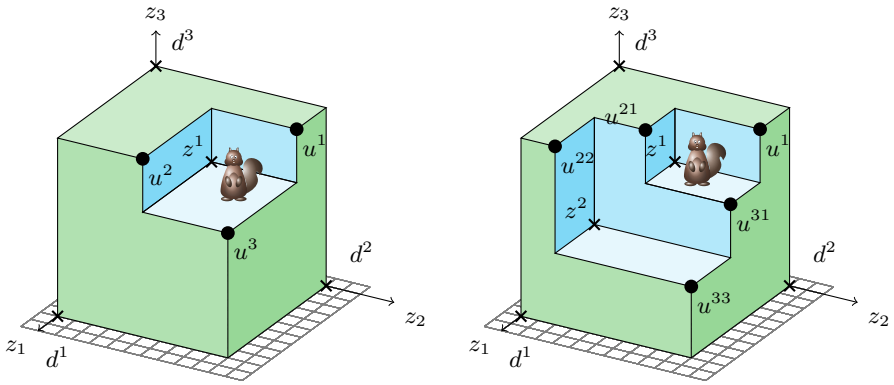


Figure 3.3: Dominated area (displayed in blue), search zones (displayed in green), and upper bounds in the tri-objective case with respect to one (left side) and two (right side) after adding the first nondominated point(s). The perspective is from above, looking towards the origin, with the dominated area appearing as sections cut out from the initial bounding box. To enhance the understanding of the view, a squirrel from the TikZlings package is added at the bottom of the cut-out area, specifically in the free space removed from the initial box.

and $u^{33} = (10, 10, 4)^\top$*, i.e.,* $U\left(\{z^1, z^2\}\right) = \{u^1, u^{21}, u^{22}, u^{31}, u^{33}\}$*, which are displayed in Figure 3.3.*

The number of local upper bounds is bounded by

$$|U(N)| = O\left(|N|^{\lfloor \frac{p}{2} \rfloor}\right) \quad \text{for } p \geq 2,$$

derived from results in algorithmic geometry Boissonnat et al. [1998] and Kaplan et al. [2008]. This bound is tight [Klamroth et al., 2015]. However, different point sets generally induce different numbers of search zones. In the case of $p = 3$ objectives and points given in general position (i.e., for each objective no pair of points shares the same value), the number of search zones is precisely equal to $2n + 1$ irrespective of the position of the given point set, see Dächert and Klamroth [2015].

3.4 Survey of Literature

This section presents a survey on algorithms for scalarization-based algorithms for multi-objective optimization problems. We focus on exact algorithms which yield the entire set of nondominated points. Moreover, while the methods in the multi-objective case are also applicable in the bi-objective case, we review methods that are explicitly designed for bi-objective problems separately. We refer to the recent survey given in Halffmann et al. [2022] and Dächert [2014] for a more comprehensive review.

3.4.1 Bi-Objective Problems

One reason for the specific role of bi-objective problems is that, in this case, Pareto optimal solutions can be sorted such that their objective values increase in one objective and decrease in the other objective. In this way, a natural ordering within the nondominated set is induced, facilitating central operations like decomposition, bound computations, and filtering for dominated solutions. This is used in the context of objective space methods, which will presented in the following.

Aneja and Nair [1979] use the weighted sum method for solving bi-objective linear programs. They focus on determining the extreme supported nondominated points saved in increasing order based on the first objective. Starting with the determination of the two lexicographic minima, the algorithm performs iterations using weighted sums, removing pairs of indices when no further extreme nondominated points exist between them. The authors show that if the bi-objective problem has k extreme nondominated points ($k \leq 2$), the algorithm performs exactly $2k-3$ iterations after having determined the lexicographic minima.

Chalmet et al. [1986] propose a similar algorithm for bi-objective integer problems, imposing constraints to eliminate known nondominated points and their dominated regions. Their approach involves solving subproblems with respect to local Nadir points. With the help of the hybrid scalarization that combines a weighted sum with appropriate constraints, it is possible to generate all supported and nonsupported nondominated points. It is shown that $2|\mathcal{Y}_N| + 1$ integer programs are solved in total. Their approach can be extended to problems with more than two objectives. We will discuss this extension in Section 3.4.2 below.

Eswaran et al. [1989] address nonlinear integer bi-objective problems using a weighted Tchebycheff method with normalized weights. The initial weight space is divided into subintervals, each boundary representing previously solved instances of weighted Tchebycheff problems. Iteratively, a subinterval is chosen and refined by determining a new weight through simple bisection. If the solution yields a new nondominated point, the current subinterval is split into two. The presented algorithm ensures the generation of all nondominated points. Similarly, Sayın and Kouvelis [2005] propose a two-stage weighted Tchebycheff method to solve bi-objective discrete optimization problems. Thereby, two variants are employed, where the first uses the Ideal point and the second uses the origin as fixed reference point. The weights are computed based on the fixed reference point and the local Nadir point of two adjacent nondominated images. Ralphs et al. [2006] propose an algorithm for integer bi-objective optimization using the (augmented) weighted Tchebycheff method. They improve the approach of Eswaran et al. [1989] by computing the weights of the subproblems based on the local Ideal and local Nadir point with respect to the two nondominated points between in which a new nondominated point is sought. The algorithm is shown to solve $2|\mathcal{Y}_N| - 1$ subproblems, where the generation of the lexicographic maxima is included and the computation of each lexicographic maximum is counted as one subproblem. Jakubovski Filho et al. [2019] expands the work of Ralphs et al. [2006] by iteratively solving Tchebycheff problems with equal weights while changing both the reference point and the bounding box with respect to previously found points.

Ulungu and Teghem [1995] address bi-objective combinatorial optimization problems using a two phase method, determining all supported nondominated points with a weighted sum scalarization. The second phase investigates triangles between adjacent supported nondominated points with a problem-specific combinatorial procedure.

Hamacher et al. [2007a] presents a-priori and a-posteriori algorithms for discrete bi-objective optimization using a lexicographic ε-constraint method. Initially, the method partitions the rectangle defined by the images of the two lexicographically optimal solutions into a lower and an upper rectangle along a horizontal axis. Subsequently, the lower rectangle undergoes a search for a nondominated point by addressing a lexicographic optimization problem. Upon identifying a nondominated point in the lower rectangle, any dominated part of the upper rectangle is discarded. Following this, the upper rectangle undergoes a similar search for a nondominated point by addressing another lexicographic optimization problem focused on the other objective. This iterative procedure is then repeated for all newly formed rectangles.

Dächert et al. [2012] explore the challenge of determining the augmentation parameter in the augmented weighted Tchebycheff problem within the bi-objective framework. The aim is to strike a balance: it should be small enough to ensure that no nondominated point is missed yet large enough to mitigate potential numerical complications.

3.4.2 Multi-Objective Problems

In the following, we describe concepts for more than two objectives. We refer again to a recent overview of exact methods Halffmann et al. [2022] and Dächert [2014]. Following Halffmann et al. [2022], we decided to split the considered articles into different categories to improve the overall presentation. While this chapter focuses on image space decomposition methods, we only consider epsilon constraint and image space decomposition methods and do not cover or only mention shortly the articles on two-phase and branch and bound methods. While norm-based methods play a minor role in the context of multi-objective minimization with more than two objectives, we do not allocate them an additional subparagraph. Additionally, although we discuss disjunctive constraint methods in Section 3.5.2.3, we do not include a separate subparagraph in this survey, referencing the relevant literature only in the corresponding section.

Methods based on Epsilon Constraint Scalarizations Epsilon constraint scalarizations lend themselves for scalarization based algorithms since they admit simple a direct subproblem formulations.

Chalmet et al. [1986] propose a recursive procedure to determine the entire set of nondominated points for integer problems in maximization format. Specifically, for the tri-objective case, they first identify all nondominated points concerning the first two objectives and that are lexicographically maximal regarding the third objective (given the computed values in the first two objectives). Among these points, the one with the minimal value in the third objective, denoted as $\bar{z}$, is identified. Subsequently, the constraint

$z_3(x) \geq \bar{z}_3 + 1$ is incorporated into the subproblem formulation. This modification allows once more the determination of all nondominated points with respect to z_1 and z_2 for the modified subproblem. Additional constraints are added to ensure that no nondominated point is computed more than once. For more than three objectives, further levels of recursion are required. The authors show that at least $p|\mathcal{Y}_N| + 1$ integer programs need to be solved.

Laumanns et al. [2006] propose a method that relies on the decomposition of the objective space into disjoint cells based on already computed nondominated points. Using a lexicographic ε-constraint method as scalarization, the search region can be projected to an $(p-1)$-dimensional subspace. In every iteration, the algorithm investigates all current cells in a specified order until a new point is obtained. They show that in the worst-case, at most $(|\mathcal{Y}_N| + 1)^{p-1}$ subproblems have to be solved in the course of the algorithm.

Özlen and Azizoğlu [2009] propose a recursive algorithm using constrainted weighted sum problems with global lower and upper bounds on the objectives, which are computed before the recursion starts. For general $p \geq 2$, a complexity of $\mathcal{O}(|\mathcal{Y}_N|^{p-1})$ with respect to the number of subproblems is derived.

The work by Özlen et al. [2014] enhances the recursive algorithm initially introduced by Özlen and Azizoğlu [2009] by incorporating information from previously solved subproblems' solutions. This entails saving the right-hand side vectors alongside either the corresponding nondominated point or indicating infeasibility for the subproblem. Prior to tackling a new subproblem with an upper bound u, the algorithm checks for the existence of a relaxation, denoted by a saved subproblem with a bound u' where $u' \leq u$. Two cases may occur: if the relaxation proves infeasible, it implies the current problem is also infeasible. Conversely, if the relaxation is feasible and all its outcomes are feasible for the current problem, then the set of outcomes for the relaxation matches those of the current problem. Alrabeeah et al. [2020] further refine this method, resulting in reduced CPU times and the number of integer programs that need to be solved. An implementation in C is presented, showcasing substantial savings of up to 95% compared to the algorithm of Özlen and Azizoğlu [2009]. Additionally, when tested on a knapsack problem with three objectives, as employed in Laumanns et al. [2005], the enhanced approach is compared to both the algorithm of Laumanns et al. [2005], revealing significant time savings in the computational time.

Kirlik and Sayın [2014] formulate a two-stage ε-constraint formulation, removing the fixed order in which the cells are investigated, the next cell selected is always the one with the largest $(p-1)$-dimensional volume between the Ideal point and the upper bound of the corresponding cell. This small change significantly reduces the number of subproblems solved. However, the theoretical worst-case bound is not improved in comparison to the approach of Laumanns et al. [2005] and Laumanns et al. [2006]. The authors implement their method in C++ using Cplex 12.4 and conduct numerical tests on knapsack and assignment problems with three and four objectives. For tri-objective problems, their approach is compared with those of Sylva and Crema [2004], Laumanns et al. [2005], and Özlen and Azizoğlu [2009]. The four-objective problem is solvable only by the authors' new method and the one of Özlen and Azizoğlu [2009]. Results demonstrate the superiority of

the proposed method over others. Additionally, for $p = 3$, the average number of solved subproblems per non-dominated point is at most 1.99. In an expansion of their earlier work, "Generating representative sets for multiobjective discrete optimization problems with specified coverage errors" [2025] introduce a new algorithm for generating representative solution sets for multiobjective discrete optimization problems, ensuring a specified coverage error. The algorithm, which builds on their method presented in Kirlik and Sayın [2014], builds on a polynomial-time method to compute an upper bound on the computationally demanding coverage error, making the approach more practical. The algorithm shows significant savings in computational effort, particularly for tri-objective problems, with performance improving almost linearly as the coverage error tolerance increases.

Generic Scalarization Based Image Space Decomposition Methods In this section, we explore criterion search methods designed to decompose the search region. With the improved efficiency of single-objective IP solvers, recent research has shifted towards image space decomposition methods, yielding algorithms that demonstrate remarkable performance in computational studies. However, as these methods exclusively operate within the image space, they often overlook valuable information embedded within the problem structure during algorithm design.

Tenfelde-Podehl [2003] proposes a recursive approach for MOCO problems. Initially, for a problem with p objectives, all corresponding $(p-1)$-objective problems are solved, potentially involving further recursion until bi-objective problems are derived. These bi-objective problems are then resolved using established methods. As per Ehrgott and Tenfelde-Podehl [2003], solving all $(p-1)$-objective problems yields a subset of the non-dominated set of the original problem, encompassing all points that define components of the Ideal and Nadir points. Using these findings, the regions that contain all remaining nondominated points may reside can be delineated after solving all $(p-1)$-objective problems. In the subsequent stage, for each nondominated point $\bar{z}$ computed in the first stage, p hyperplanes $h_i(\bar{z}) = \{z \in \mathbb{R}^p : z_i = \bar{z}_i\}$, for $i = 1, \ldots, p$, are introduced to subdivide the feasible outcome set bounded by the Ideal and nadir points into boxes. Some of these boxes can be directly excluded due to the nondominated points obtained in the first stage. All remaining boxes after this reduction are grouped based on several objectives.

In Dhaenens et al. [2010], the authors expand upon the method introduced by Tenfelde-Podehl [2003] with a three-stage procedure. The initial stage mirrors that of Tenfelde-Podehl [2003], while the second stage involves the selection of a well-dispersed subset from the points computed in the first stage. In the final stage, all remaining points are computed, resulting in an enhanced decomposition of the search region.

Dächert and Klamroth [2015] present an approach to compute the entire nondominated set of a tri-objective optimization problem. The key novelty lies in the linear dependence of the required subproblems on the number of nondominated points independent of the chosen scalarization. They develop a split criterion based on a neighborhood relation between local upper bound sets to avoid the generation of redundant boxes and to keep the number of subproblems to be solved low. This approach, initially presented for the tri-objective case, has been generalized to any number of objectives by Dächert et al. [2017]. However,

the linear dependence holds only in the bi- and tri-objective cases and increases by one with every two additional objectives. They implement their method in Matlab using Cplex 12.5 and compare it with Matlab reimplementations of Lokman and Köksalan [2013], Özlen et al. [2014], and Kirlik and Sayın [2014], all utilizing ε-constraint scalarization. For a tri-objective knapsack problem across five instances, they find that their approach consistently requires exactly $2|\mathcal{Y}_N| - 1$ subproblems. In comparison, Kirlik and Sayın [2014] requires even fewer subproblems, while Lokman and Köksalan [2013] surpasses this number in all instances, as noted in the respective papers. Additionally, Özlen et al. [2014] sometimes requires more and sometimes fewer subproblems than Dächert and Klamroth [2015].

Klamroth et al. [2015] develop a generic method to compute the entire nondominated set for any number of objectives. This approach, akin to Dächert and Klamroth [2015], utilizes local upper bounds, which are updated directly with the help of *defining points*, i.e., nondominated points which define components of local upper bounds. The authors offer variants for both the "general position" and "non-general position" cases. Assuming that the number of objectives is fixed, it is shown that the number of search zones grows only polynomially with the number of nondominated points.

Tamby and Vanderpooten [2020] expand the work of Klamroth et al. [2015] by using specific properties of the ε-constraint scalarization alongside the structural characteristics of the search region. This enables them to further decrease both the number and complexity of the subproblems that must be addressed when iteratively exploring search regions.

Boland et al. [2016] propose the *L-Shape Method* to generate all nondominated points of a tri-objective integer optimization problem. The L-Shape Search Method uses the formulation of Sylva and Crema [2004] but only consider at most one point to formulate disjunctive constraints. Geometrically, this implies that only rectangular sets and sets with a L-shape are considered, hence the name. Rectangles are explored depending on their area by searching for a nondominated point with its projection lying in the rectangle. Either a nondominated point with its projection in the rectangle is found, or it follows that no nondominated point is contained in the rectangle. In the former case, the nondominated point induces a L-shape contained in the rectangle, which is then further explored. In the latter case, the rectangle gets discarded from consideration. Since the same nondominated point might be generated more than once, the authors use enhancement strategies by checking already generated points before solving an optimization problem. Besides, the method also benefits from the use of good starting solutions for the IPs. In Boland et al. [2017a], they extend the L-shape search methodto any number of objectives. In contrast, it operates in the full-dimensional space. The authors implement their method in C++ using Cplex 12.5.1 and conduct a comparison with Özlen et al. [2014] and Kirlik and Sayın [2014]. They find that their method consistently outperforms both other approaches across the tested instances. On average, it is 32% faster than Özlen et al. [2014] and 17% faster than Kirlik and Sayın [2014]. Additionally, the number of solved IPs is reduced by 42% and 8%, respectively. Furthermore, the authors investigate the impact of their enhancements, revealing a reduction in the number of solved IPs by approximately 35%

Boland et al. [2017b] presents the so-called *quadrat shrinking method* aimed at identi-

fying the complete set of nondominated images for a tri-objective problem. The method employs the same scalarization and the same definition of boxes as Kirlik and Sayın [2014] but it diverges by altering the order of box traversal. Rather than relying on a volume-based measure, the authors adopt an iterative approach, exploring the top and right boundaries of the boxes until no new nondominated points are discovered. They present an implementation in C++ utilizing Cplex 12.6. They conduct a comparative analysis with algorithms proposed by Özlen et al. [2014], Kirlik and Sayın [2014], Boland et al. [2016], and a re-implementation of Dächert and Klamroth [2015] in C++. Across a set of tri-objective instances, they find that their method consistently outperforms all other approaches in terms of CPU time. Specifically, it is on average 39% faster than Özlen et al. [2014], 28% faster than Kirlik and Sayın [2014], 17% faster than Boland et al. [2016], and 16% faster than Dächert and Klamroth [2015].

Dächert et al. [2024] present a thorough analysis of the generic approaches of Klamroth et al. [2015] and Dächert et al. [2017] and suggest the *Defining Point Algorithm* (DPA) as a simple and efficient implementation operating fully in the objective space. The DPA keeps the number of required subproblems at a minimum by avoiding redundancies. Moreover, it offers compatibility with various scalarizations, each resulting in similarly straightforward subproblems without using disjunctive (and complicating) constraints. The implementation builds upon prior findings but presents a novel combination of them using epsilon-constraint scalarizations. They provide an implementation in C++ and use Cplex 12.9. They conduct a comparison with algorithms proposed by Özlen et al. [2014], Kirlik and Sayın [2014], and Boland et al. [2017a]. The DPA consistently outperforms all other approaches in terms of CPU time. Across Knapsack and Assignment instances with up to 5 objectives, it is on average 46% faster than Özlen et al. [2014], 54% faster than Kirlik and Sayın [2014], and 81% faster than Boland et al. [2017b]. Moreover, in high-dimensional instances with up to 10 objectives, their algorithm demonstrates superior performance in terms of CPU time compared to the other three approaches on average.

3.5 Generic Algorithms

This chapter begins by introducing a foundational, simple, and prototypical formulation of an objective space algorithm. This algorithm is based on an appropriate decomposition of the search region and the iterative solving of scalarized single-objective IPs using conventional IP solvers. The general framework of the algorithm, as outlined in Dächert et al. [2024], is provided in the pseudocode below.

The advantage of using these generic scalarization-based algorithms lies in their independence from specific problem structures. They offer flexibility by accommodating various implementations and realizations, including the sequencing of subproblems and the scalarization strategies within these subproblems. These choices directly influence how the objective space or the search region, representing the area where as-yet-undiscovered nondominated points can lie, is decomposed differently, as well as the formulation of subproblems.

Algorithm 3: Generic Scalarization-Based Algorithm

Data: MOCO problem
Result: Set of nondominated points $\mathcal{Y}_N$

```
1 Determine initial search region
2 while exists unexplored search zones of the search region do
3     Choose search zone, solve subproblem 'therein'
4     if subproblem infeasible then remove explored search zone from search region
5     else
6         save new nondominated point z^s
7         update search region based on z^s
```

Consequently, key operations in this framework include describing and updating the search region to facilitate its decomposition into *search zones*, and selecting appropriate scalarizations to formulate subproblems within the respective search zones. The previous description of the search region based in Section 3.3 was induced through the local upper bunds, while the most important scalarizations, including their descriptions, were expounded in Section 3.2.

Both decomposition and subproblem formulation significantly influence the computational efficiency of the overall method. Decomposition directly affects the number of solver calls, while the complexity of subproblems dictates the computational time required for each solver call. Notably, these two factors are interconnected, wherein the formulation of more complex subproblems can lead to substantial reductions in the search region, consequently decreasing the number of solver calls. However, this usually comes at the price of more expensive solver calls.

The method operates through iterative searches for new nondominated points. In essence, given an initial area of interest, i.e., an initial search region, each solver call results in either the generation of a new nondominated point (leading to a reduction of the search region by removing the dominated region) or the indication that the corresponding subproblem is infeasible. In the latter case, the corresponding part of the search region is excluded from further evaluations, as it does not contain any additional nondominated point.

Upon discovering and incorporating a new nondominated point into the set, the search region undergoes updates, and the search for yet-undiscovered nondominated points continues. The core of these algorithms lies in the description of the search region through its decomposition into a finite set of axis-parallel hyperrectangles known as *search zones*. Effective update operations, strategically implemented to avoid redundancies, play a crucial role in this process.

Since bi-objective problems ($p = 2$) benefit from a natural ordering, which simplifies operations such as decomposition, bound computation, and filtering of dominated points, this section begins with a focus on the bi-objective case.

3.5.1 Bi-Objective Problems

The bi-objective case ($p = 2$) exhibits several specific properties that are not present in the general multi-objective case ($p \geq 3$):

- When sorting the nondominated points in increasing (decreasing) order based on one component, they are automatically sorted decreasingly (increasingly) with respect to the other component.
- Two nondominated points can not share the same value in one component. In other words, if two points are equal in one component, at most, one of the points must be either weakly nondominated or dominated.

3.5.1.1 Algorithm Based on a ε-Constraint Scalarization

If the minimal complete nondominated set of a bi-objective problem is sought, it can be computed with the help of the ε-constraint method within $n + 1$ iterations, where n denotes the cardinality of $\mathcal{Y}_N$. More precisely, a variant of the ε-constraint method is needed since the classic ε-constraint method only yields weakly efficient solutions, as discussed in Section 3.2.

The idea of the algorithm is quite simple. By formulating appropriate ε-constraint problems, the nondominated points are generated in increasing order of one of their components. This requires the solution of n subproblems. The last subproblem ensures that no further nondominated point exists. Hence, a total of $n + 1$ subproblems are required. In the following, we describe this algorithm in more detail.

Initially, we have to fix the concrete shape of the ε-constraint problem. One design question concerns the decision which component is used for the objective function and which one for the constraint. This decision is arbitrary but has to be taken beforehand. Without loss of generality, we minimize with respect to the first component and restrict the second component in the following, which yields an ε-constraint problem of the form

$$\min_{x \in \mathcal{X}} \{z_1(x) : z_2(x) \leq \varepsilon\} \tag{3.8}$$

with $\varepsilon \in \mathbb{R}$. The second design question is related to the fact that the solution of (3.8) is only weakly efficient. As discussed in Section 3.2, various approaches ensure that the obtained solution is indeed nondominated. Since we want to solve as few IPs as possible, choosing the variant that adds an augmentation term to the objective function is preferable. Thus, we solve

$$\min_{x \in \mathcal{X}} \{z_1(x) + \rho z_2(x) : z_2(x) \leq \varepsilon\} \tag{3.9}$$

with $\varepsilon \in \mathbb{R}$, $\rho \in \mathbb{R}$. The parameter $\rho \in \mathbb{R}$ has to be chosen dependent on the problem data to avoid the solution of unnecessary subproblems. Appropriate selection requires knowledge of the bounds of the nondominated set. One possibility consists of solving the two problems

$$\min_{x \in \mathcal{X}} \{z_i(x)\} \tag{3.10}$$

for $i = 1, 2$. Let us denote the corresponding solutions by x^1, x^2. Both solutions are weakly efficient solutions. The first component of x^1 and the second component of x^2 define the Ideal point, i.e., $z^I = (z(x_1^1), z(x_2^2))^\top$.

The point composed by the other two components $(z(x_1^2), z(x_2^1))^\top$ equals the Nadir point if the solutions x^1, x^2 are nondominated. Otherwise, it is an upper bound on the nadir point that can also be used to compute ρ. Let us define $z^M = (z(x_1^2) + \delta, z(x_2^1) + \delta)^\top$ with $\delta > 0$ as upper bound on the Nadir point z^N. Then, ρ can be chosen between 0 and $(z_2^M - z_2^I)^{-1}$, e.g., as

$$\rho = \frac{1}{z_2^M - z_2^I + 1}. \tag{3.11}$$

A similar interval is constructed in Özpeynirci and Köksalan [2010].

Instead of solving the two additional IPs (3.10) to derive valid bounds on the nondominated set, one can also try to estimate bounds for the given MOCO problem directly.

After fixing the scalarization, we can formulate the algorithm, see Algorithm 4. The initial steps consist of the computation of bounds on the nondominated set as well as setting initial values for ρ and ε. Furthermore, a counter k as well as the set of generated nondomianted points S are defined. The main loop starts thereafter. In each iteration, problem (3.9) is solved with a different parameter value ε. The latter is set to the value of the second component of the previously generated nondominated point minus a value between 0 and 1. The main loop and the overall algorithm end as soon as problem (3.9) is infeasible. Note that the value of parameter ρ is set once in the beginning but is not changed afterward. It is possible to adapt ρ as well, but it is not needed, at least unless the value of ρ is not close to machine accuracy, which might cause us to run into numerical problems.

Algorithm 4: Augmented ε-constraint method for bi-objective problems

Data: BOCO problem
Result: Minimal complete set S

Solve $x^* \leftarrow \operatorname{argmin}\{z_1(x) : x \in \mathcal{X}\}$ and $x^{**} \leftarrow \operatorname{argmin}\{z_2(x) : x \in \mathcal{X}\}$
Set $z^I \leftarrow (z_1(x^*), z_2(x^{**}))^\top$ and $z^M \leftarrow (z_1(x^{**}), z_2(x^*))^\top$
Set $\rho \leftarrow \frac{1}{z_2^M - z_2^I + 1}$, $\varepsilon \leftarrow z_2^M + 1$, $k \leftarrow 1$, $S \leftarrow \emptyset$
while $\bar{x} = \operatorname{argmin}\{z_1(x) + \rho z_2(x) : z_2(x) \leq \varepsilon, x \in \mathcal{X}\}$ *has a feasible solution* $\bar{x}$ **do**
    Save solution as $x^k \leftarrow \bar{x}$, add $z^k \leftarrow z(\bar{x})$ to S
    Set $\varepsilon \leftarrow z_2^k - 0.5$
    Increase counter $k \leftarrow k + 1$
return S

Algorithm 4 computes the complete nondominated set of a bi-objective problem within $n+1$ iterations, assuming that bounds on the nondominated set are available or computed in advance. This small number of iterations is achieved because the points are generated in a sorted way, and the ε-constraint method is used as scalarization.

In Section 3.3, we introduced the concept of the search region as that part of the outcome space where undiscovered nondominated points might exist. With every generated nondominated point, the search region shrinks. In Algorithm 4, we do not explicitly state the search region since it is rather simple and consists solely of one search zone in every iteration.

The initial search region is given by the rectangle spanned by the lower bound $(z_1^I, z_2^I)^\top$ and the upper bound $(z_1^M, z_2^M)^\top$. With every generated point z^k the search region can be shrunk to the rectangle spanned by the local lower bound $l = (z_1^k, z_2^I)^\top$ and the local upper bound $u = (z_1^M, z_2^k)^\top$. Figure 3.4 displays the shrinking of the search zone during the first iteration of Algorithm 4. Thereby, the part left of z^k must be empty due to the ε-constraint method.

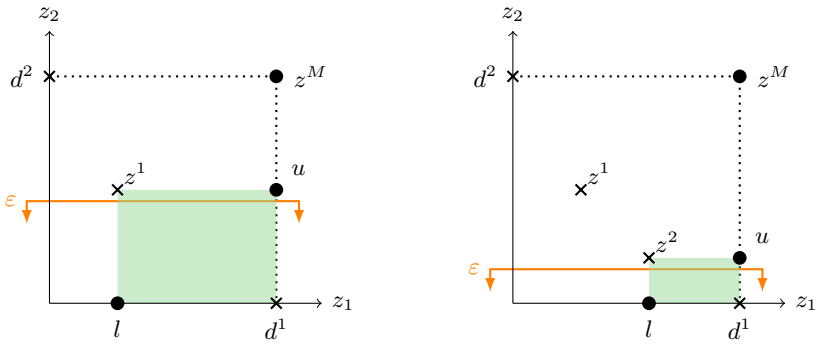


Figure 3.4: Shrinking of the search zone during the first determined non-dominates points during Procedure 4. Note that we use the origin as first lower bound for the initial region instead of the Ideal point z^I (which is dominated by the origin).

The local upper bound u, more precisely its second component, is used to define the parameter ε for the next iteration. The local lower bound l is not necessarily needed; it might remain constant throughout the whole algorithm. We only use it to define the parameter ρ in the beginning.

3.5.1.2 Generic Algorithm Independent of a Specific Scalarization

A general algorithm, independent of the ε-constraint method for scalarization, has already been formulated in Algorithm 3 and is presented in more detail below.

The procedure begins with computing an initial search region, which can be done by computing a global lower and upper bound, similar to Algorithm 4. The main loop iterates until the search region becomes empty. Initially, the search region consists of a single search zone. By setting the parameters of the chosen scalarization appropriately, the search zone is scanned for nondominated points. If the scalarization is infeasible, the

search zone is removed from the search region, and a new iteration starts. However, if a nondominated point is found, it is stored, and the search zone is updated based on that point. Specifically, the part of the search zone that is dominated by the generated point can be removed. Thus, the search zone split into two new search zones.

In the main loop, this algorithm requires, at most, $2n + 1$ subproblems, n to find each nondominated point, and $n + 1$ to detect that all search zones between adjacent nondominated points are empty.

Algorithm 3 combined with a scalarization such as the weighted Tchebycheff scalarization is particularly interesting when an incomplete representation shall be generated, i.e., if the algorithm is interrupted prematurely. Then, restricting the main loop of Algorithm 3 and Algorithm 4 to the same number of iterations, a much better dispersion of the nondominated points can be achieved by Algorithm 3 since the points are not necessarily generated in a sorted way.

Algorithm 3 can also be combined with the ε-constraint method in a very efficient way if the concrete ε-constraint method is not fixed in advance but changes in each iteration. This means that we solve $\min\{z_1(x) + \rho z_2(x) : z_2(x) \leq \varepsilon, x \in \mathcal{X}\}$ and $\min\{z_2(x) + \rho z_1(x) : z_1(x) \leq \varepsilon, x \in \mathcal{X}\}$ alternately with an appropriately formulated parameter ε. Thereby, we generate the points in a sorted way alternately from both sides. The algorithm stops "when the two ends meet", i.e., when the same solution as in the iteration before is generated. Then, the search region is empty. This approach requires the solution of $n + 1$ subproblems. Thus, it has the same complexity as Algorithm 4. The main advantage compared to Algorithm 4 is that already generated points might serve as feasible starting solutions, see Tamby and Vanderpooten [2020]. Moreover, the coverage of the nondominated set is better when the algorithm is stopped prematurely.

3.5.1.3 Disjunctive Constraints

In Algorithm 3, we save the search region as the union of rectangular search zones. In each iteration, one of the current search zones is chosen and inspected. Another idea uses the concept of disjunctive constraints to formulate the search region. By introducing additional binary variables it is possible to represent the union of the rectangular search zones explicitly. Let M_j be an upper bound on z_j for every $j = 1, 2$ and $x^1, \ldots, x^{k-1}$ denote the efficient solutions found in the previous iterations. Then, the set of constraints

$$\begin{aligned} & z_j(x) \leq \left(z_j(x^i) - 1\right) y_j^i + M_j(1 - y_j^i), && j = 1, 2,\ i = 1, \ldots, k-1, \\ & y_1^i + y_2^i \geq 1, && i = 1, \ldots, k-1, \\ & y_j^i \in \{0, 1\}, && j = 1, 2,\ i = 1, \ldots, k-1, \end{aligned} \tag{3.12}$$

describes the search region.

Figure 3.5 illustrates the impact of these constraints, along with the two binary variables for one nondominated point z^s and two nondominated points. These constraints eliminate the dominated part of z^s (z^1, z^2), leaving only the search region remaining. Due to the constraint $y_1^s + y_2^s \geq 1$, either one of the binary variables y_1^s or y_2^s must be equal to 1. If $y_1^s = 1$ and $y_2^s = 0$, only the left part of z^s is feasible; conversely, if $y_1^s = 0$ and $y_2^s = 1$, only

the lower part of z^s is feasible. When both are equal to 1, only the part of the search region remains feasible where points dominate z^s. However, as z^s represents a nondominated point, no feasible point can exist within that region. Considering this for any previously identified nondominated point z^i where $i \in \{1, \dots, k-1\}$, these combinations result in the search region being the only remaining area.

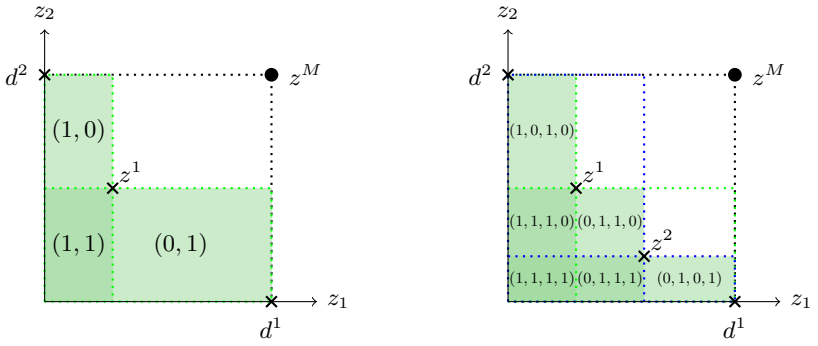


Figure 3.5: Illustration of the search zone given by the disjunctive constraints. The vectors $(y_1^1, y_2^1)^\top$ and $(y_1^1, y_2^1, y_1^2, y_2^2)^\top$ are given, respectively. Note, that we used as search zone the constraint $z_j(x) \leq (z_j(x^i))y_j^i + M_j(1 - y_j^1)$. However since we are considering a MOCO problem we even could shorten the search zone by taking $z_j(x) \leq (z_j(x^i) - 1)y_j^i + M_j(1 - y_j^1)$ as above.

These constraints are added to the scalarization. With every efficient solution, two additional binary variables, as well as, three additional constraints, have to be added to the formulation, see i.e., Klein and Hannan [1982] and Sylva and Crema [2004]. Therefore, the underlying IPs become more and more costly to be solved.

There are also hybrid methods in the literature, such as the L-shape method of Boland et al. [2016] and Boland et al. [2017a].

3.5.2 Multi-Objective Problems

The case involving three or more objectives becomes much more complex than the bi-objective case since the nondominated points cannot be sorted strictly anymore. Nevertheless, certain concepts from the previous section can still be translated.

3.5.2.1 Early approaches

As in the bi-objective case, it is still possible to generate the points in increasing order with respect to one component if the ε-constraint method or a variant of it is used as scalarization. Since the nondominated points can take any values in the remaining $p-1$ components, the choice of the right-hand side ε is no longer as clear as in the bi-objective case, where we use the value of the previously generated nondominated point.

To better understand what happens, let us begin by considering the case $p = 3$ first. Again, without loss of generality, let us generate the nondominated points in increasing

order of their first component, i.e., let us solve

$$\min_{x \in \mathcal{X}} \{z_1(x) : z_i(x) \leq \varepsilon_i,\ i = 2, \ldots, p\} \tag{3.13}$$

with $\varepsilon \in \mathbb{Z}^{p-1}$. Thus, for $p = 3$, the values of ε have to be selected from a two-dimensional grid whose values are defined by the components of previously generated nondominated points. When a point z^n is generated, we insert its second and third components into the ε-grid. The part up-right of it can be removed by the definition of nondominance. However, the part down-left can contain points, which, due to their higher first component, do not dominate z^n. Hence, further points might be found in all three cells. See Figure 3.6 for an example. The upper corner of one of these cells is used to define the ε values in the next iteration. The generated point cuts the grid into further cells. If the point lies in the part down-left from another point, it affects all three cells around it. Figure 3.6 illustrates this scenario. In total, the k nondominated points found so far cut the initial cell into $(k+1)^{p-1}$ disjoint cells. It is now crucial how to handle these cells.

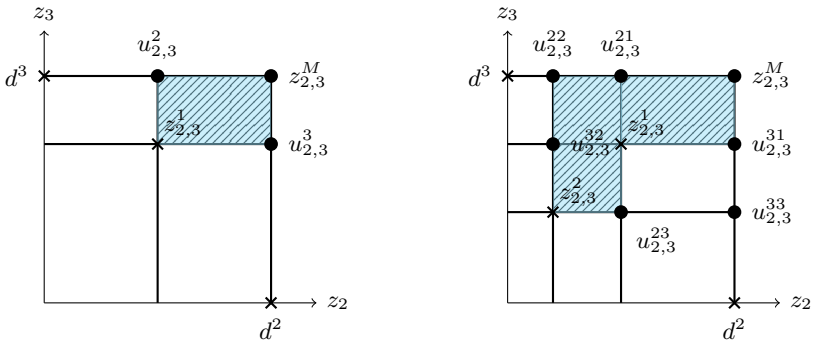


Figure 3.6: Illustration of the two-dimensional grid-projection onto the second and third objective of the three-dimensional example of Figure 3.3. While $z^2_{2,3}$ lies in the part down-left from $z^1_{2,3}$, it affects all three cells around it.

Laumanns et al. [2006] present an algorithm that turns the lower and upper edge of each of these cells into two-sided constraints of a lexicographic ε-constraint method . To the best of our knowledge it is the first algorithm that uses the components of previously generated points in an explicit and stringent way. However, it solves far too many subproblems by processing the cells one by one. Kirlik and Sayın [2014] add a simple but important detail to Laumanns et al. [2006] approach: in each iteration, they select the cell with the largest projected volume between the upper edge of a cell and the Ideal point. While this change does not improve the theoretical worst-case bound, it drastically improves its practical runtime. However, the same total number of cells is stored and slows down the algorithm in higher dimensions.

Another idea that uses the ε-constraint method as scalarization reduces the dimension of the subproblems by fixing component values until bi-objective optimization problems are obtained. In the tri-objective case, this means to fix the third component and search for all nondominated points with respect to the first two components that satisfy the bound

on the third component. The constructed bi-objective problems are solved as described in Section 3.5.1. Among the computed points, the point with the largest third component is used to update the bound on the third component. Then, a new recursion starts, computing again all nondominated points with respect to the first two components. For problems with four or more objectives, further recursions are needed. It can be proven that this recursive approach indeed finds all nondominated points of the original problem. However, a clear drawback is that the same nondominated points are typically computed in several recursive iterations. Hence, this idea requires so-called enhancement techniques that save generated nondominated points together with the bounds of the scalarization. Also, infeasible problems are saved. Before solving a subproblem, it is checked whether the solution is already known or the problem is infeasible [Chalmet et al., 1986; Tenfelde-Podehl, 2003; Özlen and Azizoğlu, 2009].

3.5.2.2 Generic Algorithm Independent of a Specific Scalarization

As described for the bi-objective case, we can also use the generic algorithm displayed in Algorithm 3 independent of a specific scalarization in the multi-objective case. What changes significantly with respect to the bi-objective case is the update of the search region. In the bi-objective case, a nondominated point lies in exactly one search zone (also called box). The latter is subdivided by the nondominated point into two new search zones. In the multi-objective case, a nondominated point might lie in multiple search zones. All these search zones have to be considered in the updating process. If we subdivide each search zone containing the current nondominated point into p new search zones, we create redundant ones. It is possible to filter out redundant search zones by pairwise comparisons [Przybylski et al., 2010b] or to prevent their creation by formulating specific criteria.

The latter is proposed in Dächert and Klamroth [2015] and Klamroth et al. [2015]. The approach of Dächert and Klamroth [2015] save a box together with its *neighboring boxes*. By comparing the components of the current nondominated point to the components of its respective neighbors, it is possible to detect redundant boxes before creating them. The approach of Klamroth et al. [2015] saves so-called *defining points* for each search zone. The comparison of components is then analogous to Dächert and Klamroth [2015], replacing the neighbors by the defining points.

Especially in the case that the nondominated points are not in general position, i.e., for each objective, no pair of points shares the same value, the algorithm of Klamroth et al. [2015] can be implemented in a much more straightforward way. While Klamroth et al. [2015] provide an algorithm with redundancies elimination with an enhanced filtering step as Przybylski et al. [2009], we focus on the presented *redundancy avoidance*, which will be described in more detail below.

Algorithm 5 is similar to Algorithm 1 from Klamroth et al. [2015], with adaption to the notation of this thesis and minor modifications similar to the Defining Point Algorithm presented in Dächert et al. [2024]. It refines the general Algorithm 3. Algorithm 6 corresponds to Algorithm 5 from Klamroth et al. [2015]. The main difficulty of Algorithm 5 applied to the multi-objective case compared to the bi-objective case is hidden in the main

loop in the line "Update $U(N)$". Algorithm 6 shows this update in the non-general case. It will be described in more detail below.

Definition 3.6. *For an local upper bound $u \in U(N)$ all components $u_i, i = 1, \dots, p$ are induced by a corresponding z_i^s of a point $z^s \in N$ that satisfies $z_i^s = u_i$ and $z_j^s \leq u_j$ for all $j \in \{1, \dots, p\} \backslash \{i\}$. We call such a point* defining point *for component i of the local upper bound u.*

Moreover, once we identify at least one nondominated point z within the interior of the bounding box (typically achieved after solving the initial subproblem), the global upper bound $(M, \dots, M)^T$ is dominated, and each local upper bound contains at least one defining point that is not a dummy point.

In a broader context, whenever a new nondominated point $\bar{z}$ is discovered within $U(N)$, the set $U(N)$ necessitates an update: for all $U(N)$ with $\bar{z} < u$, u is replaced by p smaller local upper bounds $u^1, \dots, u^p$. These bounds are defined as follows:

$$u_j^i = \begin{cases} \bar{z}_j, j = i \\ u_j, j \neq i \end{cases} \quad \text{for } i = 1, \dots, p.$$

In essence, to compute u^i, the i-th component of u is substituted with the value of the nondominated point $\bar{z}$ in the i-th component while leaving all other components unchanged. Thus, all components of a local upper bound u result from previously generated upper bounds for $p - 1$ components and from the currently added point $\bar{z}$ for the remaining component. This ensures that $u^i \leqq u$ for all $i = 1, \dots, p$, effectively reducing the search region. For clarity, we refer to the resulting local upper bound u^i as an i-child of u. It is worth noting that $\bar{z} < u$ may hold true for multiple local upper bounds in $U(N)$. In such scenarios, some of the generated i-children may be redundant for describing $U(N \cup \bar{z})$ and can thus be eliminated. The process of identifying redundant local upper bounds is extensively detailed in works by Dächert and Klamroth [2015], Klamroth et al. [2015], and Dächert et al. [2017]. They also outlined efficient procedures for iteratively updating the set $U(N)$ upon the addition of further nondominated points to N.

Algorithm 6 uses defining points to avoid the existence of redundant search zones. Before going into the details of the procedure, we state some findings from Klamroth et al. [2015]. Sine we added dummy points in the initial set of nondominated points N, any component value of a local upper bound is defined by a point in N.

Observe that a dummy point d^j can only define the j-th component of any local upper bound, which equals M. Given that no point from $\mathcal{Y}_N$ falls below or equals m on any component, m can not serve as a component value of a local upper bound. Therefore, and since M is unique in the component values of a dummy point, d^j stands as the only dummy point capable of defining component j.

For any $z \in \mathbb{R}^p$, let z_{-j} denote the $(p-1)$-dimensional vector comprising all components of z excluding component j, for a given $j \in \{1, \dots, p\}$. Furthermore, for any $z, a \in \mathbb{R}^p$ and any $j \in \{1, \dots, p\}, (z_j, a_{-j})$ represents the vector $(a_1, \dots, a_{j-1}, z_j, a_{j+1}, \dots, a_p)^\top$, referred to as the j-th projection of vector z onto vector a. Note that, in the following of

this chapter, we do not distinguish between column and row vectors, and thus omit the transpose notation for simplicity.

The subsequent proposition gives valuable property to those points that define each component of a local upper bound.

Proposition 3.7 (Klamroth et al., 2015). *For any local upper bound $u \in U(N)$ and $j \in \{1, \dots, p\}$, there exists $z \in \hat{N}$ such that $z_j = u_j$ and $z_{-j} < u_{-j}$.*

Consequently, we can define those points that define local upper bounds.

Definition 3.8 (Klamroth et al., 2015). *For any local upper bound $u \in U(N)$, we denote by $Z^j(u) = \{z \in \hat{N} : z_j = u_j \text{ and } z_{-j} < u_{-j}\}$ the set of defining points of u for component $j, j \in \{1, \dots, p\}$.*

Rather than computing all projections, we can now precisely characterize the projections that are kept in the set P to circumvent redundancies.

Theorem 3.9 (Klamroth et al., 2015). *Let N be the set of already found nondominated points, and let $\bar{z}$ be the newly found nondominated point of $\mathcal{Y}_N$. Consider a local upper bound $u \in U(N)$ such that $\bar{z} < u$. Let $z_j^{\max}(u) = \max_{k \neq j} \min\{z_j : z \in Z^k(u)\}$. Then, for any $j \in \{1, \dots, p\}$, $(\bar{z}_j, u_{-j})$ is a local upper bound of $U(N \cup \{\bar{z}\})$ if and only if $\bar{z}_j > z_j^{\max}(u)$*

According to Theorem 3.9, we can avoid redundancies by keeping track of the p points that define each local upper bound and only generating the projections of $\bar{z}$ satisfying the conditions of Theorem 3.9. The corresponding algorithm is detailed in Algorithm 6. Note that each component of the vector $z^{\max}(u)$ for a given local upper bound u is utilized at most once in all iterations of Algorithm 6. Hence, it is computed only prior to its use, namely in Step 7. Moreover, this vector is not sufficient to compute the vector $z^{\max}\left(u^j\right)$ associated with a local upper bound u^j defined from u. Indeed, it is required to keep track of all points that define the component values of u^j, as is done in Steps 12-14.

Example 3.10 (Klamroth et al., 2015). *To illustrate the concept of defining points and redundancy avoidance, reconsider the example out of Figure 3.3, which is displayed again in Figure 3.7. The points that are added are $z^1 = (3, 5, 7)^\top$ and $z^2 = (6, 2, 4)^\top$. First it holds $U(\emptyset) = \{(M, M, M)^\top\}$. The defining points are given by d^1, d^2, d^3. At the first iteration, z^1 yields the three local upper bounds, namely $u^1 = (3, M, M)^\top$, $u^2 = (M, 5, M)^\top$, and $u^3 = (M, M, 7)^\top$ so that $U\left(\{z^1\}\right) = \{u^1, u^2, u^3\}$. The points that define the local upper bounds are*

$$\begin{array}{lll} Z^1\left(u^1\right) = z^1, & Z^2\left(u^1\right) = d^2, & Z^3\left(u^1\right) = d^3, \\ Z^1\left(u^2\right) = d^1, & Z^2\left(u^2\right) = z^1, & Z^3\left(u^2\right) = d^3, \\ Z^1\left(u^3\right) = d^1, & Z^2\left(u^3\right) = d^2, & Z^3\left(u^3\right) = z^1. \end{array}$$

and $z^{\max}\left(u^1\right) = (m, 5, 7), z^{\max}\left(u^2\right) = (3, m, 7)$, and $z^{\max}\left(u^3\right) = (3, 5, m)$. Then at the second iteration, the point $z^2 = (6, 2, 4)$ strictly dominates u^2 and u^3 and we have:

$$\begin{array}{lll} z_1^2 > z_1^{\max}\left(u^2\right), & z_2^2 > z_2^{\max}\left(u^2\right), & z_3^2 \leq z_3^{\max}\left(u^2\right) \\ z_1^2 > z_1^{\max}\left(u^3\right), & z_2^2 \leq z_2^{\max}\left(u^3\right), & z_3^2 > z_3^{\max}\left(u^3\right), \end{array}$$

thus we obtain again the four new local upper bounds $u^{21} = \left(z_1^2, u_{-1}^2\right), u^{22} = \left(z_2^2, u_{-2}^2\right)$, $u^{31} = \left(z_1^2, u_{-1}^3\right)$, and $u^{33} = \left(z_2^2, u_{-2}^3\right)$, i.e., $U\left(\{z^1, z^2\}\right) = \{u^1, u^{21}, u^{22}, u^{31}, u^{33}\}$.

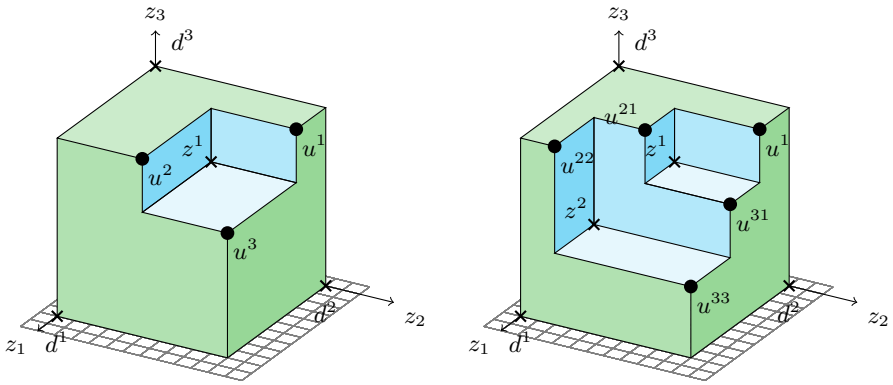


Figure 3.7: Tri-objective Example.

Worst-case Complexity Decomposing the search region $S(N)$ into $|U(N)|$ pairwise non-redundant search zones allows for solving a single-objective subproblem for each corresponding zone, aiming to either discover new nondominated points or determine the absence of further nondominated points within each zone. Consequently, the cardinality of $U(N)$ dictates the number of subproblems to solve, impacting the computational complexity of Algorithm 5. Therefore, in Algorithm 5 the cardinality of $U(N)$ decisive the number of subproblems that need to be solved.

Theorem 3.11. *The number of local upper bounds is bounded by*

$$|U(N)| = O(|N|^{\lfloor \frac{p}{2} \rfloor}) \quad \text{for } p \geq 2.$$

The proof can be derived from results of Boissonnat et al. [1998] and Kaplan et al. [2008]. This bound is tight [Klamroth et al., 2015]. However, different point sets generally induce different numbers of search zones. In the case of $p = 3$ objectives and points given in general position (i.e., for each objective, no pair of points shares the same value), the number of search zones is precisely equal to $2n + 1$ irrespective of the position of the given point set, see Dächert and Klamroth [2015]. The connection between *local upper bounds* and their *defining points* can be utilized to derive relationships between search zones and subproblems. Utilizing this information towards coordinated search zone selection and subproblem formulation leads to significant speed-ups in scalarization-based algorithms [Tamby, 2018].

A scalarization method is considered *suitable* if it either discovers a new nondominated point or identifies the respective search zone as empty (thereby excluding it from further consideration) during the solution of an appropriate scalarization within a search zone. Notably, as seen in Section 3.2, this property is fulfilled by widely used methods such as the ε-constraint method and the weighted Tchebycheff method. Thus, when the number of objectives p is fixed and the natural decomposition is combined with a suitable scalarization method, Theorem 3.11 indicates a polynomial bound on the total number of iterations of Algorithm 5. The following theorem and proof is presented in Dächert et al. [2024].

Theorem 3.12 (Dächert et al., 2024)**.** *When the natural decomposition of the search region into non-redundant search zones is combined with an suitable scalarization method, the total number of iterations of Algorithm 5 can be bounded by*

$$\mathcal{O}\left(|\mathcal{Y}_N|^{\lfloor \frac{p}{2} \rfloor}\right) \quad \text{for } p \geq 2,$$

which is polynomial in the size of the nondominated set $|\mathcal{Y}_N|$ when p is fixed.

Proof. Due to Theorem 3.11 in each iteration, the procedure only investigates at most $\mathcal{O}(k^{\lfloor \frac{p}{2} \rfloor})$ search zones, where k is the number of already determined nondominated points. Thus, if the complete set of nondominated points $\mathcal{Y}_N$ is determined, the procedure has to investigate at most $\mathcal{O}(|\mathcal{Y}_N|^{\lfloor \frac{p}{2} \rfloor})$ remaining search zones to prove that they are empty. If one of these empty search zones is detected as empty in an earlier iteration of the procedure it is removed and not changed in a later iteration. Indeed, only those search zones are updated and split that contain a newly detected nondominated point, and this situation can not occur when the considered search zone is empty. Therefore, the number of iterations can be bounded by

$$\mathcal{O}\left(|\mathcal{Y}_N| + |\mathcal{Y}_N|^{\lfloor \frac{p}{2} \rfloor}\right) = O\left(|\mathcal{Y}_N|^{\lfloor \frac{p}{2} \rfloor}\right) \quad \text{for } p \geq 2.$$

□

We now consider the complexity of Algorithm 6. Initially, the approach computes the set A of local upper bounds whose associated search zones contain $\bar{z}$. This involves $|U(N)|$ dominance tests if $U(N)$ is maintained as a simple linked list. Subsequently, we consider $p|A|$ candidate locate upper bounds. The values $z_j^{\max}(u)$ need to be computed just prior to they are needed, each of which takes constant time. Furthermore, updating the references to the p points that define each local upper bound takes constant time for each new upper bound. Step 14 entails considering at most $|N|$ points in a set $Z^k(u)$. This yields a worst-case complexity of $\mathcal{O}(|N||A|)$.

To the best of our knowledge, this approach stands as the only objective space method with a provably polynomial bound on the number of required solver calls while maintaining simplicity in these solver calls, without relying on disjunctive constraints and/or additional integer variables.

Reduction of Search Zone when Using ε-Scalarization After addressing the update of the search region upon discovering a new nondominated point, our next step is to examine the selection of the search region (Line 3 of Algorithm 5). Making use of a specific property of the ε-constraint method and incorporating it into Line 4 of Algorithm 5, we can present a selection criterion outlined in Dächert et al. [2024] to further refine the search region. Instead of choosing the search zone arbitrarily but according to some criterion, it allows the removal of a newly created search zone directly in every iteration where a new nondominated point is determined. Here fore, we need the notion of *neighbor* from Definition 3.6 in Dächert et al. [2017], as defined below:

Algorithm 5: Defining Point Algorithm (similar to Algorithm 1 from Klamroth et al. [2015])

Data: MOCO problem
Result: Set of nondominated points $\mathcal{Y}_N$

$[z^I, z^M] \leftarrow$ ObtainBounds // Compute Ideal point and global upper bound
$u \leftarrow$ CreateInitialBox(z^M) // Initialize u
$N \leftarrow \emptyset$; $U(N) \leftarrow \{u\}$ // Initialize lists of upper bounds U and nondominated points N
while $U(N) \neq \emptyset$ **do**
    Select $u \in U(N)$
    Solve chosen scalarization within the search zone $C(u)$
    if *problem is feasible* **then**
        Let $\bar{z}$ be an optimal point
        $N \leftarrow N \cup \{\bar{z}\}$ // save new nondominated point
        Update $U(N)$ by Algorithm 6 // update search region
    else
        $U(N) \leftarrow U(N) \setminus \{u\}$ // remove explored search zone from search region
return $\mathcal{Y}_N = N$

Definition 3.13 (Dächert et al., 2024)**.** *Two local upper bounds $u, u' \in U(N)$ are called* neighbors *if they share $p-1$ defining points, among which exactly one changes its position from some $j \in \{1, \dots, p\}$ to some $k \in \{1, \dots, p\}, k \neq j$. That is, there are two indices j, k with $j \neq k$ such that $z^j(u) = z^k(u')$ while $z^i(u) = z^i(u')$ for all $i \in \{1, \dots, p\} \backslash \{j, k\}$. We then say that u is a j-neighbor of u', and that u' is a k-neighbor of u.*

With this definition of neighbors, we can present the following minimum component selection criterion.

Theorem 3.14. *If, in every iteration of Algorithm 5, a search zone $\bar{u}$ is selected, which is minimal in some component $i \in \{1, \dots, p\}$, i.e., for which*

$$\bar{u}_i = \min \{u_i : u \in U(N)\}$$

for some $i \in \{1, \dots, p\}$ holds, and if we then solve an ε-constraint method , then, whenever there is a solution $\bar{z} \in \mathcal{Y}_N$ with $\bar{z} < \bar{u}$, the i-child of $\bar{u}$ exists, i.e., $\bar{u}^i \in U(N \cup \{\bar{z}\})$. Moreover, $\bar{u}^i$ can not contain further nondominated points, i.e., $\left\{z \in \mathcal{Y}_N : z < \bar{u}^i\right\} = \emptyset$.

Consequently, we can remove one search zone per iteration when a new nondominated point is discovered.

Another selection criterion is presented in Tamby and Vanderpooten [2020], a search zone u^* is selected by computing

Algorithm 6: Algorithm 5 from Klamroth et al. [2015]

Data: $U(N)$ together with $Z^j(u)$ for all $j \in \{1,\dots,p\}$, $\bar{z}$
// Set of local upper bounds and associated defining points, new point
Result: $U(N \cup \{\bar{z}\})$

$A \leftarrow \{u \in U(N) : \bar{z} < u\}$
// Search zones that contain $\bar{z}$
$P \leftarrow \emptyset$
for $u \in \mathcal{U}(\mathcal{N})$ *and* $j \in \{1,\dots,p\}$ *such that* $\bar{z}_j = u_j$ *and* $\bar{z}_{-j} < u_{-j}$ **do**
  $Z^j(u) \leftarrow Z^j(u) \cup \{\bar{z}\}$
for $u \in A$ **do**
  for $j \in \{1,\dots,p\}$ **do**
    $z_j^{\max}(u) \leftarrow \max_{k \neq j} \min\{z_j : z \in Z^k(u)\}$
    Check for the condition of Theorem 3.9
    if $\bar{z}_j > z_j^{\max}(u)$ **then**
      $P \leftarrow P \cup \{u^j\}$
      Let $u^j \leftarrow (\bar{z}_j, u_{-j})$
      $Z^j(u^j) \leftarrow \{\bar{z}\}$ **for** $k \in \{1,\dots,p\} \setminus \{j\}$ **do**
        $Z^k(u^j) \leftarrow \{z \in Z^k(u) : z_j < \bar{z}_j\}$
$U(N \cup \{\bar{z}\}) \leftarrow (U(N) \setminus A) \cup P$

$$(i^*, u^*) = \begin{cases} (1, (M, \dots, M)) & \text{if } N = \emptyset, \\ \arg\max\{h(i,u) : u \in U(N), i \in \{1,\dots,p\}, u_i \neq M\} & \text{otherwise,} \end{cases} \tag{3.14}$$

where

$$h(i,u) = \prod_{j \neq i} \left(u_j - z_j^I\right)$$

The index i^* serves to make use of starting solutions. Note that we could additionally choose the index i, which then determines the index to be optimized in the ε-constraint method in the first approach. Then, previously generated nondominated points can be used as starting solutions.

We can now formulate the selection criterion presented by Tamby and Vanderpooten [2020], which selects the search zone with the largest projected hypervolume. Note that, except in the first iteration, a product consisting of $p-1$ terms has to be computed for every search zone with $u_i \neq M$.

Theorem 3.15 (Dächert et al., 2024)**.** *If, in every iteration of Algorithm 5, a search zone $\bar{u}$ according to 3.14 is selected, and if we then solve an ε-constraint method , then, whenever there is a solution $\bar{z} \in \mathcal{Y}_N$ with $\bar{z} < \bar{u}$, the i-child of $\bar{u}$ exists, i.e., $\bar{u}^i \in U(N \cup \{\bar{z}\})$. Moreover, $\bar{u}^i$ can not contain further nondominated points, i.e., $\left\{z \in \mathcal{Y}_N : z < \bar{u}^i\right\} = \emptyset$.*

This demonstrates that the selection criterion of Tamby and Vanderpooten [2020] possesses the same beneficial property as the selection criterion of Dächert et al. [2024]. However, the selection criterion of Dächert et al. [2024] is simpler since it is only based on one component value of a search zone and does not require any computations, such as the projected hypervolumes.

3.5.2.3 Disjunctive Constraints and Alternative Search Strategies

In our previous chapters, we emphasized the formal description of the search region and its subdivision into search zones, implying decomposition-based algorithms. While objective space methods can be distinguished in their search for nondominated outcome vectors within the objective space, particularly within the search region, we now want to examine alternative search strategies.

In MOCO problems, disjunctive programming can be used when defining the search region as a union of rectangular sets and combining all parts of the feasible set by 'or' statements. Disjunctive constraints are often reformulated using a big-M approach alongside binary variables to activate or deactivate parts of the feasible set. By considering all parts simultaneously, a new nondominated point can be found by solving only one integer programming problem. However, as new nondominated points emerge, the computational complexity increases due to additional constraints and binary variables.

As in the bi-objective case, we could formulate the search region with disjunctive constraints as in Sylva and Crema [2004] or their extension in Sylva and Crema [2007]. By introducing additional binary variables it is possible to represent the union of the rectangular search zones explicitly. Let M_j be an upper bound on z_j for every $j = 1, \dots, p$ and $x^1, \dots, x^{k-1}$ denote the efficient solutions found in the previous iterations. Then, the set of constraints

$$\begin{aligned}
&z_j(x) \leq \left(z_j(x^i) - 1\right) y_j^i + M_j(1 - y_j^i), && j = 1, \dots, p,\ i = 1, \dots, k-1, \\
&\sum_{j=1}^{p} y_j^i \geq 1, && i = 1, \dots, k-1, \\
&y_j^i \in \{0, 1\}, && j = 1, \dots, p,\ i = 1, \dots, k-1,
\end{aligned} \tag{3.15}$$

describes the search region. These constraints are added to the scalarization. With every efficient solution, two additional binary variables, as well as, three additional constraints, have to be added to the formulation. Therefore, the underlying IPs become more and more costly to be solved.

Klein and Hannan [1982] use the disjunctive constraints to simultaneously consider all remaining parts of the search region. In each iteration with $s \geq 1$ non-dominated points already known, the authors solve a problem by incorporating $p - 1$ disjunctive constraints

$$\bigwedge_{i=1}^{s-1} \left(\bigvee_{\substack{j=1,\dots,p \\ j \neq k}} z_j(x) \leq z_j(x^i) - \delta_j \right).$$

Here, an arbitrarily chosen objective z_k, where $k \in \{1, \ldots, p\}$, is minimized.

Sylva and Crema [2004] revisit the concept introduced by Klein and Hannan [1982], altering the objective by employing a weighted sum to avoid weakly nondominated points. Additionally, they reformulate the disjunctive constraints using binary variables. However, with each iteration, this approach adds p binary variables and $p + 1$ constraints, which makes this approach computationally demanding. In fact, Sylva and Crema [2004] only generate complete representations for the bi-objective case. For three objectives, they limit the numerical study to the generation of in-complete representations. This approach is refined in Sylva and Crema [2007] where the binary variables are derived directly from the disjunctive optimization problem and are optimized independently. As a result of this refinement, the computational time of the approach is significantly reduced compared to their earlier work in 2004.

Lokman and Köksalan [2013] present two algorithms. The first algorithm extends the method proposed by Sylva and Crema [2004] by incorporating all other objectives scaled by a small constant directly into the objective function. This adjustment justifies the omission of one constraint and one binary variable in each iteration. However, the algorithm still suffers from the issue of a rapidly growing number of constraints and binary variables. Conversely, the second algorithm capitalizes on the insight that for every feasible point, at most one constraint from the disjunctive formulation suffices for each of the p–1 objectives. Given s nondominated points, a set of $ns + 1$ ε-constraint problems with augmentation terms are solved in the next iteration. The specific values on the right-hand side depend on components derived from previously determined nondominated points. Consequently, this approach can also be considered a method based on epsilon-constraint scalarizations as discussed above. The theoretical upper bound on the number of subproblems is $\mathcal{O}(|\mathcal{Y}_N|^{p-1})$. They demonstrate that their second approach surpasses the algorithms of Sylva and Crema [2004], their own enhancement of it, and the algorithm of Özlen and Azizoğlu [2009]. Furthermore, for $m = 3$, an average of 2.13 subproblems are solved per nondominated point across randomly generated instances of multi-objective knapsack, minimum spanning tree, and shortest path problems.

In Bektas [2018] a closer examination of the disjunctive constraints formulated by Klein and Hannan [1982] reveals that only certain conjunctions of disjunctive constraints need to be considered. Others that lead to dominated sets of inequalities can be excluded via a filtering step in the proposed algorithm. The computational experiments demonstrate that their prosposed algorithm is competitive with the method by Kirlik and Sayın, 2014 for smaller instances, but its relative performance improves as the number of objectives increases or when the problem is more constrained.

Several methods incorporate recursion to reduce the dimensionality of the objective space and, hence, the search region until bi- or single-objective problems are obtained. This recursive approach can be executed in various ways. One method involves selecting two objectives and computing all nondominated points relative to these two objectives while iteratively narrowing the upper bounds of other objectives in each recursion level. This is the basic idea of Chalmet et al. [1986] and Özlen and Azizoğlu [2009] and Özlen et al. [2014]. Alternatively, some methods compute all possible combinations of recursions.

They tackle all p corresponding $(p-1)$-dimensional optimization problems and so on, creating a tree with bicriteria optimization problems in its leaf nodes. It has been demonstrated by Ehrgott and Skriver [2003] that the points obtained by solving all $(p-1)$-criteria problems represent a subset of the nondominated set of the original problem. This recursive strategy has been employed, for instance, in Tenfelde-Podehl [2003] and Dhaenens et al. [2010] and Przybylski et al. [2010b]. However, a drawback of recursive algorithms is that nondominated points are typically computed several times since they are often optimal for multiple recursive subproblems. Given the costly nature of solving such problems, performance must avoid the repetition of already known nondominated points.

Hybrid methods combine ideas from the aforementioned categories. For instance, Boland et al. [2016] propose the L-shape search method for tri-objective problems, integrating disjunctive constraints and decomposition. The idea is to use disjunctive constraints only concerning the lastly generated nondominated point, which results in an L-shape element that is investigated with priority and shrunk quickly towards the Ideal point. Unexplored rectangular sets are saved for later investigation during the algorithm. Boland et al. [2017a] extend this approach to any number of objectives. Similarly, the method proposed by Boland et al. [2017b] for tri-objective optimization problems merge recursion and decomposition. It deals with upper-bound vectors similar to other decomposition approaches. However, for a certain sequence of problems to be solved, it keeps the bound on one of the objectives fixed, as in a recursive method.

3.5.3 Examples

In this part, we want to use the p-objective *assignment problem (pAP)*, a special case of the minimum cost flow problem, to illustrate the methods presented above. The assignment problem can be formally defined as:

$$\begin{aligned}
\min z_k(x) = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}^k x_{ij} & & k = 1,\dots,p \\
\sum_{i=1}^{n} x_{ij} &= 1 & j = 1,\dots,n \\
\sum_{j=1}^{n} x_{ij} &= 1 & i = 1,\dots,n \\
x_{ij} &\in \{0,1\} & i,j = 1,\dots,n.
\end{aligned}$$

where all objective function coefficients c_{ij}^k are non-negative integers and x is the $n \times n$ matrix of decision variables with $x_{ij} = 1$ if task i is assigned to agent j, and 0 otherwise. In our examples, we consider three-dimensional assignment problems, i.e. the case $n = 3$. Note that the feasible set then consists of six feasible elements, namely the six possible permutations of $\{1, 2, 3\}$, each representing a feasible assignment of tasks to agents. These assignments can be written as:

$$(1,2,3),\ (1,3,2),\ (2,1,3),\ (2,3,1),\ (3,1,2),\ (3,2,1),$$

where the i-th entry in the permutation indicates the agent j assigned to task i. For instance, the permutation $(2,3,1)$ means that task 1 is assigned to agent 2, task 2 to agent 3, and task 3 to agent 1. Thus in this example, it holds $x_{12} = 1, x_{23} = 1, x_{31} = 1$, and $x_{11}, x_{13}, x_{21}, x_{22}, x_{32}, x_{33} = 0$.

We start by illustrating Algorithm 4 on an instance of a three-dimensional assignment problem with two objectives, i.e. $p = 2$.

Example 3.16. *For this, consider the following cost matrices*

$$C^1 = \begin{pmatrix} 1 & 3 & 1 \\ 3 & 2 & 1 \\ 4 & 2 & 0 \end{pmatrix}, \text{ and } C^2 = \begin{pmatrix} 4 & 1 & 4 \\ 2 & 2 & 1 \\ 0 & 1 & 2 \end{pmatrix}.$$

Algorithm 4 first computes $x^* = argmin\{z_1(x) : x \in \mathcal{X}\}$ *and* $x^{**} = argmin\{z_2(x) : x \in \mathcal{X}\}$, *obtaining the points* $z(x^*) = (3,8)^\top$ *and* $z(x^{**}) = (8,2)^\top$. *Consequently, we derive* $z^I = (3,2)^\top$ *and* $z^M = (8,8)^\top$, *which define the first zone of interest, as shown in Figure 3.8. Note that we set* $\varepsilon = z_1^M + 1 = 9$ *in the first iteration since it is unclear whether* $(3,8)^\top$ *is a nondominated solution at this stage.*

In the first iteration of the while loop (lines 4-7) in Algorithm 4, we identify the point $z^1 = (3,8)^\top$ *as a nondominated point. For the next iteration, we set* $\varepsilon = 7.5$ *and solve for* $\bar{x} = \arg\min_{x \in \mathcal{X}}\{z_1(x) + \rho z_2(x),\ z_2(x) \leq \varepsilon\}$, *obtaining the second nondominated point* $z^2 = (4,6)^\top$. *Note that* ρ *was chosen as* $\rho = \frac{1}{z_2^M - z_2^I + 1} = \frac{1}{8 - 7.5 + 1} = 0.4$.

Continuing with $\varepsilon = 5.5$, *we find two more nondominated points* $z^3 = (6,5)^\top$ *and* $z^4 = (8,2)^\top$ *in subsequent iterations. This process yields the complete set of nondominated points* $S = \{z^1, z^2, z^3, z^4\}$. *The first three iterations are illustrated in Figure 3.8.*

Among the six feasible solutions, four are nondominated and two are dominated. The corresponding permutations (feasible assignments) and their objective vectors are presented in the table below.

Permutation	**Assignment (nonzero x_{ij})**	$z(x)$	**Label**
(1,2,3)	x_{11}, x_{22}, x_{33}	$z^1 = (3,8)$	nondominated
(1,3,2)	x_{11}, x_{23}, x_{32}	$z^2 = (4,6)$	nondominated
(2,1,3)	x_{12}, x_{21}, x_{33}	$z^3 = (6,5)$	nondominated
(2,3,1)	x_{12}, x_{23}, x_{31}	$z^4 = (8,2)$	nondominated
(3,1,2)	x_{13}, x_{21}, x_{32}	$(6,7)$	dominated
(3,2,1)	x_{13}, x_{22}, x_{31}	$(7,6)$	dominated

Table 3.1: Objective values and dominance status for all 6 feasible assignments in the 3x3 bi-objective assignment problem.

In order to illustrate Procedure 5, we add a third objective function.

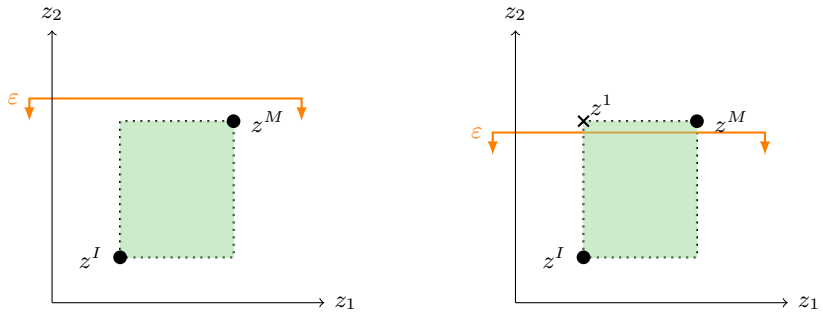


Figure 3.8: The fist two iterations of Algorithm 4 on the Instance of Example 3.16.

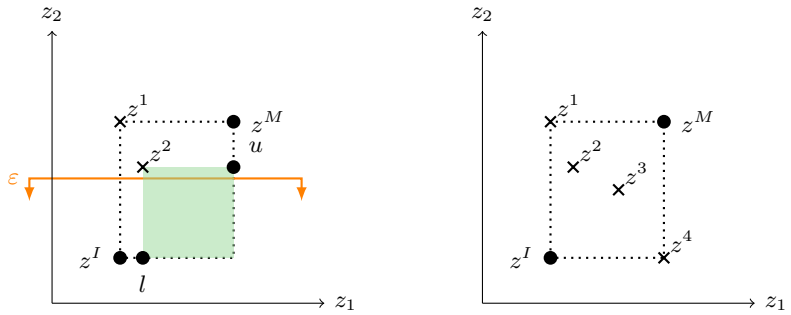


Figure 3.9: The first three iterations of Algorithm 4 on the Instance of Example 3.16, and the final nondominated set.

Example 3.17. *Consider the following instance of the assignment problem with three objectives*

$$C^1 = \begin{pmatrix} 1 & 3 & 1 \\ 3 & 2 & 1 \\ 4 & 2 & 0 \end{pmatrix}, \; C^2 = \begin{pmatrix} 4 & 1 & 4 \\ 2 & 2 & 1 \\ 0 & 1 & 2 \end{pmatrix}, \text{ and } \; C^3 = \begin{pmatrix} 3 & 4 & 1 \\ 0 & 3 & 1 \\ 2 & 1 & 2 \end{pmatrix}.$$

We start with the initial box. To keep the illustrations simple, we use the origin and the point $z^M = (10, 10, 10)^\top$ *as the initial starting box, which is slightly bigger than the box obtained from* z^I *and* z^M*. Therefore, we have three dummy points* $d^1 = (M, 0, 0)^\top$, $d^2 = (0, M, 0)^\top$, *and* $d^3 = (0, 0, M)^\top$, *and* $U(N) = \{u = (10, 10, 10)^\top\}$ *before starting the while loop. In this example, we choose the* ε*-constraint method with minimization of the first objective. Then, the first iteration yields the point* $z^1 = (3, 8, 8)^\top$*. We obtain* $N = \{z^1\}$ *and* $U(N) = \{u^1, u^2, u^3\}$*. Figure 3.10 shows how the upper bound set is updated.*

However, after the selection criterion Theorem 3.14, u^1 *cannot contain any further nondominated point and thus may be omitted from further investigation. Hence, for the next iteration* u^2 *may be selected, and an* ε*-constraint scalarization minimizing the first objective over* $C(u^2)$ *can be applied to determine* $z^2 = (4, 6, 5)^\top$*. It holds* $z^2 < u^2$, *and* $z^2 < u^3$, *therefore* u^2 *and* u^3 *need to be updated. Figure 3.11 displays again the updating of the upper bound set. Again* u^{21} *cannot contain further nondominated points after Theorem 3.14*

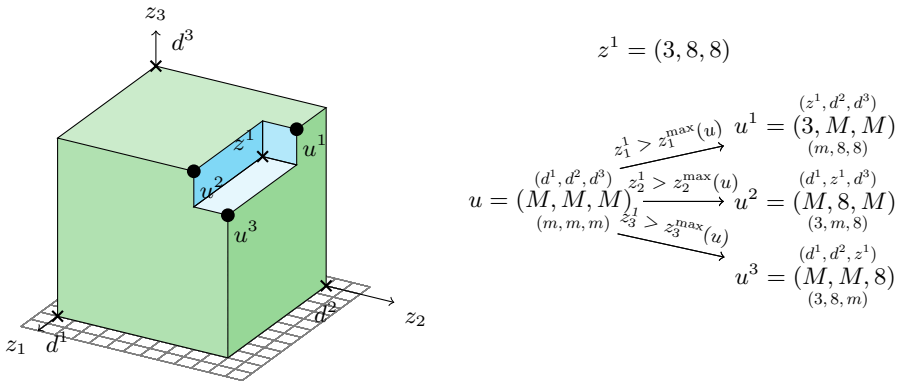


Figure 3.10: Updating of the upper bound set after adding z^1 to N. Above an local upper bound u the set of defining points $(Z^1(u), Z^2(u), Z^3(u))$ are shown, while below the vector $z^{\max}(u)$ is displayed. We omit transposed signs to simplify the notation.

and thus can be neglected. It holds $N = \{z^1, z^2\}$ *and* $U(N) = \{u^1, u^{21}, u^{22}, u^{31}, u^{33}\}$. *Note*

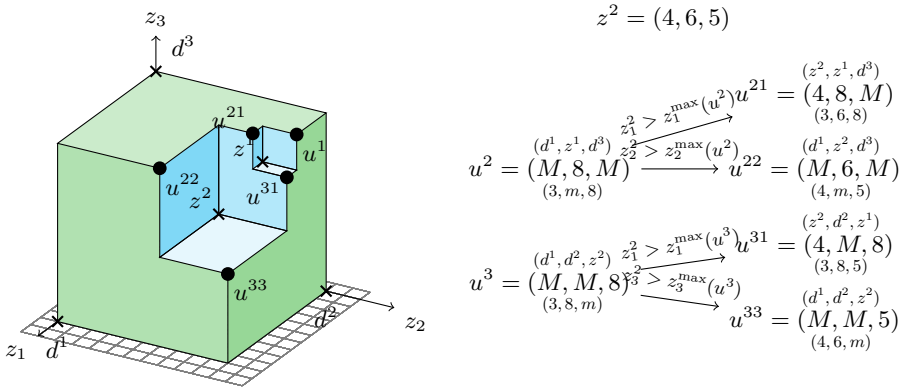


Figure 3.11: Updating of the upper bound set after adding z^2 to N.

that for the sake of completeness, we include the upper bounds u^1 *and* u^{21}. *However, as established after the selection criterion, the corresponding search zones do not contain any further nondominated points. In the following, all upper bounds shown in parentheses share this property.*

In the third iteration of the while loop, $z^3 = (6, 5, 6)^\top$ *is detected, and it only holds* $z^3 < u^{22}$ *and therefore only* u^{22} *needs an updating. We have* $N = \{z^1, z^2, z^3\}$ *and* $U(N) = \{u^1, u^{21}, u^{221}, u^{222}, u^{223}, u^{31}, u^{33}\}$. *In the subsequent iteration,* $z^4 = (8, 2, 7)^\top$ *is determined. Only* $u^{222} = (M, 5, M)$ *needs an update. Therefore,* $N = \{z^1, z^2, z^3, z^4\}$ *and* $U(N) = \{u^1, u^{21}, u^{221}, u^{2221}, u^{2222}, u^{2223} u^{223}, u^{31}, u^{33}\}$. *After that, the nondominated point* $z^5 = (6, 7, 2)^\top$ *is found and only* u^{33} *needs an update. Afterwards, no other nondominated point exists for any of the search zones determined by the local upper bounds. Hence, the complete set of nondominated points for the given assignment problem is given*

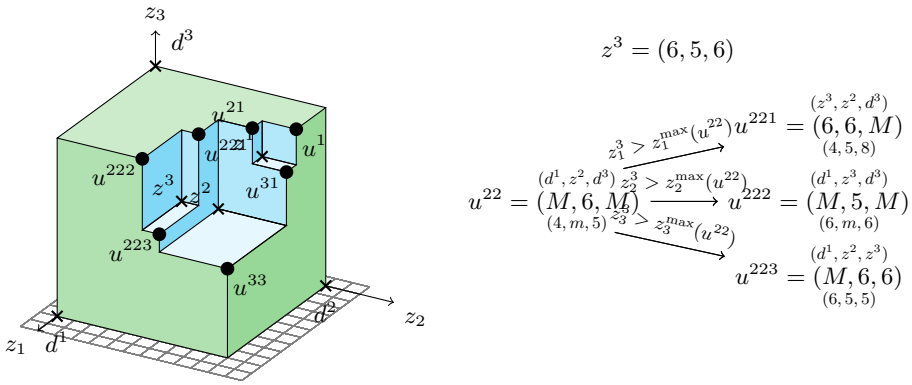


Figure 3.12: Updating of the upper bound set after adding z^3 to N.

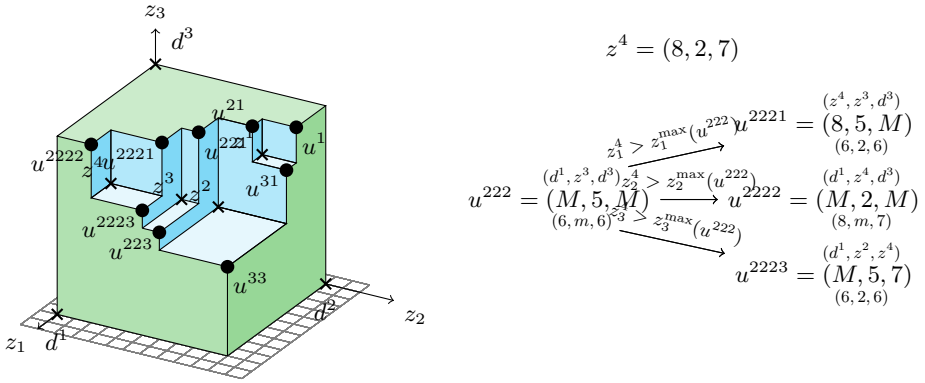


Figure 3.13: Updating of the upper bound set after adding z^4 to N.

by $N = \{z^1, z^2, z^3, z^4, z^5\}$. *Thus, by introducing the third objective, one of the previously*

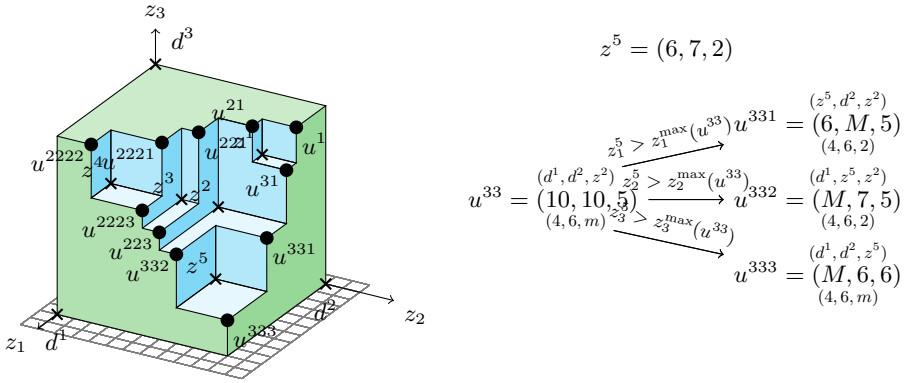


Figure 3.14: Updating of the upper bound set after adding z^5 to $\mathcal{N}$.

dominated solutions in Example 3.16 becomes nondominated in the tri-objective setting.

To illustrate the variability in the number of local upper bounds induced by different sets

Permutation	**Assignment (nonzero x_{ij})**	$z(x)$	**Label**
(1,2,3)	x_{11}, x_{22}, x_{33}	$z^1 = (3, 8, 8)$	nondominated
(1,3,2)	x_{11}, x_{23}, x_{32}	$z^2 = (4, 6, 5)$	nondominated
(2,1,3)	x_{12}, x_{21}, x_{33}	$z^3 = (6, 5, 6)$	nondominated
(2,3,1)	x_{12}, x_{23}, x_{31}	$z^4 = (8, 2, 7)$	nondominated
(3,1,2)	x_{13}, x_{21}, x_{32}	$z^5 = (6, 7, 2)$	nondominated
(3,2,1)	x_{13}, x_{22}, x_{31}	$(7, 6, 6)$	dominated

Table 3.2: Objective vectors and dominance status for all 6 feasible assignments in the 3×3 tri-objective assignment problem.

of nondominated points with the same cardinality in higher dimensions, we will consider two sets of three four-dimensional nondominated points.

Example 3.18. *Consider the sets of nondominated points* $N^1 = \{z^1 = (3, 8, 8, 2), z^2 = (4, 6, 5, 3), z^3 = (6, 5, 6, 7)\}$ *and* $N^2 = \{z^1, z^2, z^4\}$ *with* $z^4 = (6, 5, 9, 1)$. *The induced local upper bounds (after applying a redundancy avoidance step) for these sets are as follows: For* N^1, *the local upper bounds are:* $U(N^1) = \{u^1 = (3, M, M, M), u^{21} = (4, 8, M, M), u^{221} = (6, 6, M, M), u^{222} = (M, 5, M, M), u^{223} = (M, 6, 6, M), u^{224} = (M, 6, M, 7)$, $u^{24} = (M, 8, M, 3), u^{31} = (4, M, 8, M), u^{33} = (M, M, 5, M), u^{34} = (M, M, 8, 3), u^4 = (M, M, M, 2)\}$. *This results in* $|U(N^1)| = 11$ *local upper bounds. Without redundancy avoidance, there would be two additional local upper bounds* u^{23} *and* u^{32}.

However for N^2, *the local upper bounds are:* $U(N^2) = \{u^1 = (3, M, M, M), u^{21} = (4, 8, M, M), u^{221} = (6, 6, M, M), u^{222} = (M, 5, M, M), u^{223} = (M, 6, 9, M)$, $u^{241} = (6, 8, M, 3), u^{243} = (M, 8, 9, 3), u^{31} = (4, M, 8, M), u^{33} = (M, M, 5, M), u^{34} = (M, M, 8, 3), u^{41} = (6, M, M, 2), u^{43} = (M, M, 9, 2), u^{44} = (M, M, M, 1)$ $\}$. *This results in* $|U(N^2)| = 13$ *local upper bounds. Without redundancy avoidance, there would be four additional local upper bounds:* u^{224}, u^{242}, u^{244}, *and* u^{42}. *Therefore, we have* $|U(N^1)| < |U(N^2)|$.

The number of intermediate local upper bounds in Procedure 5 can vary depending on the order in which the nondominated points are added, or in which the respective search zones are explored. For instance, while it holds that $|U(N^1 \cup \{z^4\})| = |U(N^2 \cup \{z^3\})| = 14$, *adding the points in the order* $N^1 = \{z^1, z^2, z^3\}$ *results in 11 local upper bounds, whereas adding the points in order* N^2 *results in 13 bounds, as shown before.*

3.6 Conclusion

This chapter presented the most common scalarization methods, which play an important role in multi-objective optimization. These methods use the strengths of single-objective solvers by transforming the multi-objective problem into a single-objective problem. Various scalarization techniques exist, each differing in complexity and the quality of the solution set obtained. While the weighted-sum method, one of the oldest and more straightforward scalarizations, offers simplicity but may only determine the set of supported efficient solutions, more complex scalarizations like the ε-constraint or weighted

Tchebycheff method can find every efficient solution for multi-objective combinatorial optimization problems.

The focus extended to the description of the search region, particularly in its natural decomposition in a finite set of axis-parallel hyperrectangles known as search zones. By combining scalarization methods with coordinated search zone selection, generic scalarization-based approaches that offer flexibility and applicability are introduced. These algorithms operate through iterative searches for new nondominated points, dynamically updating the search region.

For multi-objective problems, we highlight the defining point algorithm introduced in Klamroth et al. [2015], which effectively updates the search region upon discovering new nondominated points to eliminate redundancies.

Integrating this algorithm with suitable scalarization methods, such as those presented in Tamby and Vanderpooten [2020] or Dächert et al. [2024], yields a versatile approach to computing the nondominated set of multi-objective integer programming problems. This approach provides a competitive balance between the number of required solver calls and the numerical complexity of each call, with the number of solver calls bounded by a polynomial in the number of nondominated points in the worst case. At the same time, subproblems are kept simple by using the ε-constraint method or weighted Tchebycheff method. Comparative analyses presented in Dächert et al. [2024] demonstrate its clear superiority in terms of CPU time over other state-of-the-art implementations of generic scalarization-based algorithms presented in Kirlik and Sayın [2014] and Özlen et al. [2014] and Boland et al. [2017a].

Furthermore, they explored the performance differences between the ε-constraint method and the weighted Tchebycheff method. While the augmented ε-constraint method proves superior when generating the entire nondominated set due to its reduced number of integer programming problems, the weighted Tchebycheff method may be more advantageous when generating only a subset of the nondominated set, known as an incomplete representation.

Additionally, the chapter discussed other search strategies, such as disjunctive constraints and hybrid approaches, outlining their advantages and limitations. Overall, this chapter underscores the importance of generic approaches in solving multi-objective optimization problems and explores various implementation possibilities.

As computational time increases with the number of objectives, future research should focus on developing parallel variants of the defining point algorithm. Parallel variants based on other algorithms have been proposed by Pettersson and Ozlen [2019b] and Turgut et al. [2019], and it remains to be seen how a parallel defining point algorithm competes against these approaches.

MOCO belongs to the class of computationally intractable problems [Ehrgott, 2005]. Most MOCO problems involve a huge set of nondominated points, resulting in high computational effort, particularly for high-dimensional problems or large instances. The non-supported nondominated points, which lie in the interior of the upper image, often outnumber those on the nondominated frontier (see, e.g., Visée et al. 1998). Consequently,

researchers frequently focus on identifying representative subsets, such as the supported nondominated points.

The following chapters of this thesis present results on the representation, definition, and computation of supported nondominated points, with a particular emphasis on MOIMCF problems.

4 On Supportdness in Multi-Objective Combinatorial Optimization

This chapter addresses an inconsistency in various definitions of supported nondominated points within multi-objective combinatorial problems (MOCO). MOCO problems are known to contain supported and nonsupported nondominated points, with the latter typically outnumbering the former. Supported nondominated points are, in general, more straightforward to determine, can serve as representations, and are used in two-phase methods to generate the entire nondominated point set. Despite their importance, several different characterizations for supported efficient solutions (and supported nondominated points) are used in the literature.

While these definitions are equivalent for multi-objective linear problems, they can yield different sets of supported nondominated points for MOCO problems. We show by an example that these definitions are not equivalent for MOCO or general multi-objective optimization problems. Moreover, we analyze the structural and computational properties of the resulting sets of supported nondominated points. These considerations motivate us to summarize equivalent definitions and characterizations for supported efficient solutions and to introduce a distinction between *supported* and *weakly supported* efficient solutions.

The result of this chapter is joint work with Michael Stiglmayr and is available as a technical report Könen and Stiglmayr [2025b], which has been accepted for publication in the Journal of Multi-Criteria Decision Analysis.

4.1 Introduction

The set of efficient solutions and the set of nondominated points are often decomposed into the sets of supported and nonsupported efficient solutions and nondominated points, respectively. Supported solutions are typically defined as solutions that can be obtained as optimal solutions of a weighted sum scalarization. The computation of nonsupported nondominated points requires different scalarization techniques and is often computationally more expensive. Moreover, for MOCO problems, the nonsupported nondominated points typically outnumber the supported ones, as observed in Visée et al. [1998]. The determination of supported nondominated points has gained attention for several reasons. First, these points are generally easier to determine than the nonsupported nondominated points. Second, they can serve as a foundation for the second phase of *two-phase methods*, which aims to generate the entire nondominated point set using information derived from the supported nondominated points (see, e.g., Pasternak and Passy 1973; Visée et al. 1998; Hamacher et al. 2007b; Przybylski et al. 2008; Eusébio and Figueira 2009b; Przybylski et al. 2010b; Dai and Charkhgard 2018). Recently, a computational study Sayın

[2024] showed that the set of supported nondominated points can be used as a high-quality representation in multi-objective discrete optimization problems, focusing specifically on binary knapsack and assignment problems. In Chapter 5, we show that this also holds for network flow problems.

Several studies focus on identifying or analyzing the supported or extreme supported nondominated point set across various combinatorial problems, including bi- and multi-objective integer network flow problems [Eusébio and Figueira, 2009b; Medrano and Church, 2015; Raith and Sedeño-Noda, 2017; Könen and Stiglmayr, 2025a; Könen and Stiglmayr, 2023], bi- and multi-objective minimum spanning trees [Sourd et al., 2006; Silva and Clímaco, 2007; Correia et al., 2021], bi- and multi-objective shortest path problems [Edwin Romeijn and Smith, 1999; Sedeño-Noda and Raith, 2015; Medrano and Church, 2014], bi- and multi-objective combinatorial unconstrained problems [Bökler, 2018; Schulze et al., 2019], bi- and multi-objective knapsack problems [Visée et al., 1998; Argyris et al., 2011; Schulze, 2017], and bi- and multi-objective assignment problems [Tuyttens et al., 2000; Gandibleux et al., 2003; Przybylski et al., 2010b]. Additional work on general MOCO problems can be found in Gandibleux et al. [2001], Jesus [2015], and Sayın [2024], among others.

Beyond MOCO problems, there is research on the identification of supported solutions in multi-objective mixed integer problems [Özpeynirci and Köksalan, 2010; Pettersson and Ozlen, 2019a; Bökler et al., 2024] or addressing supportedness in non-convex problems [The Luc, 1995; Liefooghe et al., 2014; Liefooghe et al., 2015]. Summarizing, supported nondominated points are often more straightforward to determine, can serve as high-quality representations, and can be used in two-phase methods to generate the entire nondominated point set.

However, despite their importance, several characterizations exist for supported efficient solutions and analogously for supported nondominated points. These different definitions differ in the literature and sometimes even within a single publication. While these definitions are equivalent in the case of multi-objective linear problems [Isermann, 1974; Ehrgott, 2005], they can lead to different sets of supported efficient solutions and thus supported nondominated point sets.

Supported nondominated points for MOCO problems are often characterized as nondominated points that lie on the boundary of the upper image and that they only lie on the *nondominated frontier*, whereas nonsupported solutions are characterized as nondominated points that lie in the interior of the upper image [Eusébio and Figueira, 2009b; Przybylski et al., 2010a]. However, depending on the definition, nonsupported nondominated points may exist that lie on the boundary of the upper image or supported nondominated points that lie on *weakly nondominated faces* and thus do not lie on the nondominated frontier. In this case, they cannot be obtained as optimal solutions of a weighted sum problem with weights strictly greater than zero.

This motivates us to distinguish between *supported* efficient solutions and *weakly supported* efficient solutions. An efficient solution is denoted as weakly supported if it is an optimal solution of a weighted sum scalarization with non-negative weights. In contrast to the definition of supportedness, weakly supportedness allows single weights to have

a value of zero. Note that the terminology weakly supported efficient solutions, despite the fact that these solutions are still efficient, is motivated by geometric considerations: any weakly supported nondominated point that is not a supported nondominated point lies on a weakly nondominated face, but not on a nondominated face. In contrast, supported points necessarily lie on nondominated faces. A nonsupported efficient solution is then defined as an efficient solution that is neither supported nor weakly supported. Then the following characterizations for MOCO problems hold: While weakly supported nondominated points lie on the boundary of the upper image, supported nondominated points lie only on the nondominated frontier, i. e., only on maximally nondominated faces. The nonsupported nondominated points lie in the interior of the upper image. We will present an example where the set of supported efficient solutions is a proper subset of all weakly efficient solutions in Section 4.2. The clear distinction between the sets of supported nondominated and weakly supported nondominated solutions is also necessary, as the corresponding problems may differ in their output time complexity. In particular, in the case of the multi-objective integer minimum cost flow problem, Chapter 7 shows that supported efficient solutions can be determined in *output-polynomial time*, whereas this is not the case for the *weakly supported solutions* unless $\mathbf{P} = \mathbf{NP}$. Note that the computation of the weakly supported solutions is as "hard" as the determination of all nondominated points.

While the above characterization of supported nondominated points applies to MOCO problems, they do not extend to general MOO problems, as illustrated in Example 3.14 in Ehrgott [2005]. This distinction arises because the weighted sum scalarization method with weights strictly positive is only capable of identifying supported nondominated points that are also properly nondominated in the sense of Geoffrion [Geoffrion, 1968], as discussed in Ehrgott [2005]. For MOCO problems, the distinction between efficient and properly efficient solutions vanishes since every efficient solution is also properly efficient [Ehrgott, 2005]. However, for general MOO problems, the set of properly nondominated points may form a strict subset of the nondominated points on the nondominated frontier, highlighting the limitations of the weighted sum approach in capturing the entire set of supported solutions in general MOO problems [Ehrgott, 2005].

This chapter focuses on the supportedness in multi-objective discrete problems. However, further investigation into the supportedness of general MOO problems remains sparse and presents an important area for future research. A recent study in Chlumsky-Harttmann [2025] provided a first approach toward a categorization of supportedness definitions for general MOO problems.

The remainder of the chapter is structured as follows. In Section 4.2, the different definitions and characterizations found in the literature are presented. A counterexample is given that shows that these definitions are not equivalent for MOCO or general nonconvex multi-objective optimization problems. Additionally, it discusses the distinction between supported and weakly supported nondominated points.

4.2 Supported and Weakly Supported Nondominated Points

Several definitions and characterizations of supported nondominated points exist in the literature.

Definition 4.1 (Conflicting Definitions of Supportedness)**.** *A point $y' \in \mathcal{Y}$ is called supported*

(1) if y' is nondominated and y' lies on the boundary of the upper image $\mathcal{Y}^{\geqq} := \operatorname{conv}(\mathcal{Y}_N + \mathbb{R}^p_{\geqq})$, i.e., $y' \in \mathcal{Y}_N \cap \partial\mathcal{Y}^{\geqq}$ [Eusébio and Figueira, 2009a; Eusébio and Figueira, 2009b; Liefooghe et al., 2014; Liefooghe et al., 2015; Correia et al., 2021]. The corresponding set of supported nondominated points is denoted by $\mathcal{Y}_{S\partial}$.

(2) if y' is nondominated and y' is located on the nondominated frontier *defined as the set $\{y \in \operatorname{conv}(\mathcal{Y}_N) \colon \operatorname{conv}(\mathcal{Y}_N) \cap (y - \mathbb{R}^p_{\geqq}) = \{y\}\}$ [Hamacher et al., 2007b], the set supported nondominated points according to this definition is denoted by $\mathcal{Y}_{SNF}$.*

(3) if y' is the image of a supported efficient solution, which are those efficient solutions that can be obtained as optimal solutions of a weighted sum scalarization with weights strictly greater zero, i.e. y is an image of a solution $x \in \arg\min P_\lambda$ for a $\lambda \in \Lambda_p$ [Visée et al., 1998; Ehrgott, 2005; Raith and Ehrgott, 2009; Przybylski et al., 2010a; Argyris et al., 2011; Raith and Sedeño-Noda, 2017]. The set of supported nondominated points obtained with respect to this definition is denoted by $\mathcal{Y}_{S\lambda}$.

Note that some publications adopt the definition related to $\mathcal{Y}_{S\lambda}$ but allowing $\lambda \in \Lambda_p^0$, provided that y is nondominated, see, e.g. Gandibleux et al. [2001] and Liefooghe et al. [2015]. Regarding definition $\mathcal{Y}_{SNF}$, Hamacher et al. [2007b] uses the wording *efficient frontier* instead of the nondominated frontier. There also exist definitions based on convex combinations of the nondominated points as given in Özpeynirci and Köksalan [2010]. Furthermore, note that $\mathcal{Y}_{SNF}$ equals $\mathcal{Y}_S$ from Definition 2.49.

Isermann [1974] showed that for a MOLP, the set of nonsupported efficient solutions coincides with the set of optimal solutions of the weighted sum method with weights strictly greater than zero, i.e., in MOLP, the weighted sum scalarization achieves completeness when varying λ within Λ_p. Consequently, for a MOLP, it holds $\mathcal{Y}_{S\partial} = \mathcal{Y}_{SNF} = \mathcal{Y}_{S\lambda}$.

Theorem 4.2. *For a MOLP it holds $\mathcal{Y}_{S\partial} = \mathcal{Y}_{SNF} = \mathcal{Y}_{S\lambda}$.*

Proof. According to Theorem 2.46, any nondominated point $y' \in \mathcal{Y}$ in MOLP can be obtained as the image of an optimal solution $x \in \arg\min P_\lambda$ with $\lambda \in \Lambda_p$, and with Theorem 2.44 we obtain $\mathcal{Y}_{S\lambda} = \mathcal{Y}_N$. Any point $\hat{y} \in \mathcal{Y}_{S\partial}$ or $\hat{y} \in \mathcal{Y}_{SNF}$ is nondominated per definition and therefore $\hat{y} \in \mathcal{Y}_{S\lambda}$. It follows

$$\mathcal{Y}_{S\lambda} \supseteq \mathcal{Y}_{S\partial} \text{ and } \mathcal{Y}_{S\lambda} \supseteq \mathcal{Y}_{SNF}.$$

Note that for an MOLP, $\mathcal{Y}$ is polyhedral and thus $\mathcal{Y} = \operatorname{conv}(\mathcal{Y})$. Consider $\hat{y} \in \mathcal{Y}_{S\lambda}$, i.e., $\hat{y}$ is an image of an optimal solution $x' \in \arg\min P_\lambda$ with $\lambda \in \Lambda_p$. Since $\mathcal{Y}$ is polyhedral, $\hat{y}$ must lie in a face of $\operatorname{conv}(\mathcal{Y})$ [Nemhauser and Wolsey, 1999]. Hence, $\hat{y} \in \partial \operatorname{conv}(\mathcal{Y})$

and from Theorem 2.44 it follows $\hat{y} \in \mathcal{Y}_N$. Thus, it follows $\hat{y} \in \partial \operatorname{conv}(\mathcal{Y}_N) \cap \mathcal{Y}_N$, and hence $\hat{y} \in \mathcal{Y}_\partial$. Furthermore, since $\operatorname{conv}(\mathcal{Y}_N)$ is a nonempty convex set, it holds that $\hat{y}$ is a minimal element of $\operatorname{conv}(\mathcal{Y}_N)$ induced by the cone $-\mathbb{R}^p_{\leqq}$ [Boyd and Vandenberghe, 2004] and we can conclude that $((\hat{y} - \mathbb{R}^p_{\leqq})\backslash\hat{y}) \cap \operatorname{conv}(\mathcal{Y}_N) = \varnothing$. Consequently, it follows that $\hat{y} \in \mathcal{Y}_{SNF}$. Together it holds,

$$\mathcal{Y}_{S\lambda} \subseteq \mathcal{Y}_{S\partial} \text{ and } \mathcal{Y}_{S\lambda} \subseteq \mathcal{Y}_{SNF}.$$

This concludes $\mathcal{Y}_{S\partial} = \mathcal{Y}_{SNF} = \mathcal{Y}_{S\lambda}$. □

However, this equivalence does not hold for the discrete or the non-linear case. In the literature, supported nondominated points for MOCO are often characterized as nondominated points on the boundary of the upper image and that they only lie on the maximally nondominated faces, i.e., the nondominated frontier. In contrast, the nondominated are characterized as nondominated points that lie in the interior of the upper image, e.g., [Eusébio and Figueira, 2009b; Przybylski et al., 2010a; Correia et al., 2021].

Suppose a supported vector is defined according to Definition 4.1 (2) and (3), i.e., considering the sets $\mathcal{Y}_{S\lambda}$ and $\mathcal{Y}_{SNF}$. In that case, nondominated points may exist on the boundary of the upper image. Conversely, if a supported nondominated point is defined according to (1), i.e., points contained in $\mathcal{Y}_{S\partial}$, there may exist supported nondominated points on the boundary of the upper image which are not lying on the nondominated frontier, i. e., which lie on weakly nondominated faces. This inconsistency motivates us to develop new, consistent definitions of supported and weakly supported nondominated points.

Definition 4.3 (Supported/Weakly Supported)**.** *An efficient solution is called a* weakly supported efficient solution *if it is an optimal solution of a weighted-sum scalarization P_λ for some weight $\lambda \in \Lambda_p^0$. Moreover, if the weight is strictly positive $\lambda \in \Lambda_p$ it is called* supported efficient solution*. The corresponding image is called* weakly supported *or* supported (nondominated) vector*, respectively.*

Note that the terminology weakly supported efficient solutions, despite the fact that these solutions are still efficient, is motivated by geometric considerations: any weakly supported nondominated point that is not a supported nondominated point lies on a weakly nondominated face, but not on a nondominated face. In contrast, supported nondominated points necessarily lie on nondominated faces.

Let $\mathcal{Y}_S$ and $\mathcal{Y}_{wS}$ be the set of all supported nondominated points and the set of all weakly supported nondominated points, respectively. Accordingly, $\mathcal{X}_S$ ($\mathcal{X}_{wS}$) is the set of all (weakly) supported efficient solutions. We denote the cardinality of $\mathcal{X}_S$ by $|\mathcal{X}_S| = \mathcal{S}$. With these definitions, the following characterizations for MOILPs hold: While weakly supported nondominated points lie on the boundary of the upper image, supported nondominated points lie only on the nondominated frontier, i. e., only on maximally nondominated faces. The unsupported nondominated points lie in the interior of the upper image.

Theorem 4.4. $\mathcal{Y}_{wS} = \mathcal{Y}_{S\partial}$

Proof. We follow the proofs of Theorem 3.4 and Theorem 3.5 in Ehrgott [2005] with slight modifications. To prove the equality of the two sets, we show that the subset relation holds in both directions.

- $\mathcal{Y}_{wS} \subseteq \mathcal{Y}_{S\partial}$: Let $\hat{y} \in Y_{wS}$, i.e., there exists a $\lambda \in \Lambda_p^0$ such that $\hat{y}$ is an image of an optimal solution of (P_λ) and $\hat{y}$ is nondominated.

  Suppose that $\hat{y} \notin \mathcal{Y}_{S\partial}$, i.e., $\hat{y} \notin \partial \operatorname{conv}(\mathcal{Y}_N + \mathbb{R}^p_{\leqq})$. Then $\hat{y}$ must lie in the interior of $\operatorname{conv}(\mathcal{Y}_N + \mathbb{R}^p_{\leqq})$. Thus, there exists an ε-neighborhood $B(\hat{y}, \varepsilon)$ of $\hat{y}$, defined as $B(\hat{y}, \varepsilon) := \hat{y} + B(0, \varepsilon) \subset \operatorname{conv}(\mathcal{Y}_N + \mathbb{R}^p_{\leqq})$, where $B(0, \varepsilon)$ is an open ball with radius ε centered at the origin. Let $d \in \mathbb{R}^p_{>}$. Then we can choose some $\alpha \in \mathbb{R}$, $0 < \alpha < \varepsilon$ such that $\alpha\, d \in B(0, \varepsilon)$. Now, $y' = \hat{y} - \alpha\, d \in \operatorname{conv}(\mathcal{Y}_N + \mathbb{R}^p_{\leqq})$ with $y'_k < \hat{y}_k$ for $k = 1, \ldots, p$ and

  $$\sum_{k=1}^{p} \lambda_k\, y'_k < \sum_{k=1}^{p} \lambda_k\, \hat{y}_k,$$

  because at least one of the weights λ_k must be positive. This contradiction implies the result.

- $\mathcal{Y}_{S\partial} \subseteq \mathcal{Y}_{wS}$: Let $\hat{y} \in \mathcal{Y}_{S\partial} \implies \hat{y} \in \partial \operatorname{conv}(\mathcal{Y}_N + \mathbb{R}^p_{\leqq})$ and $\hat{y}$ is nondominated, hence it exists a supporting hyperplane $\{y \in \mathbb{R}^p : \lambda^\top y = \lambda^\top \hat{y}\}$ with $\lambda \in \mathbb{R}^p \setminus \{0\}$ and it holds $\lambda^\top y \geq \lambda^\top \hat{y}$ for all $y \in \operatorname{conv}(\mathcal{Y}_N + \mathbb{R}^p_{\leqq})$. Furthermore, by applying the separation theorem to the disjoint sets $\{y + d \colon d \in \mathbb{R}^p_{>}\}$ and $\{\hat{y} - d' \colon d' \in \mathbb{R}^p_{>}\}$, we obtain:

  $$\lambda^\top (y + d - \hat{y}) \geq 0 \geq \lambda^\top (-d')$$

  for all $y \in \operatorname{conv}(\mathcal{Y}_N + \mathbb{R}^p_{\leqq})$, and $d, d' \in \mathbb{R}^p_{>}$. Choose $d' = e_k + \varepsilon\, e$, where e_k is the k-th unit vector, $e = (1, \ldots, 1)^\top \in \mathbb{R}^p$. With $\varepsilon > 0$ arbitrary small we see that $\lambda_k \geq 0$ for all $k = 1, \ldots, p$. Thus, $\hat{y}$ is the image of an optimal solution to P_λ with $\lambda \in \Lambda_0^p$ and is nondominated. It follows $\hat{y} \in \mathcal{Y}_{wS}$.

□

Theorem 4.5. *For MOCO problems, it holds* $\mathcal{Y}_S = \mathcal{Y}_{SNF}$.

Proof. $\mathcal{Y}_S \subseteq \mathcal{Y}_{SNF}$: Let $\hat{y} \in \mathcal{Y}_S = \mathcal{Y}_{S\lambda}$, i.e., $\hat{y}$ is the image of an optimal solution of P_λ with $\lambda \in \Lambda_p$. Since $\operatorname{conv}(\mathcal{Y}_N)$ is a nonempty convex set, $\hat{y}$ is a minimal element of $\operatorname{conv}(\mathcal{Y}_N)$ induced of the cone $-\mathbb{R}^p_{\leqq}$ and hence it holds $((\hat{y} - \mathbb{R}^p_{\leqq}) \setminus \hat{y}) \cap \operatorname{conv}(\mathcal{Y}_N) = \varnothing$ [Boyd and Vandenberghe, 2004]. Consequently, it follows that $\hat{y} \in \mathcal{Y}_{SNF}$.

$\mathcal{Y}_S \supseteq \mathcal{Y}_{SNF}$: Any point $\hat{y} \in \mathcal{Y}_{SNF}$ is a minimal element of the set $\operatorname{conv}(Y_N)$ induced by the cone $-\mathbb{R}^p_{\leqq}$. Thus $\hat{y}$ is a nondominated point for the set $\operatorname{conv}(\mathcal{Y}_N)$. If we define the relaxation MOLP of the given MOCO problem by

$$\min_{x \in \operatorname{conv}(\mathcal{X})} C\, x,$$

it follows that $\hat{y}$, is an image of an optimal solution for the reduced MOLP. According to Theorem 2.46, there is a $\lambda \in \Lambda_p$ such that $\hat{y}$ is the image of an optimal solution to P_λ and hence $\hat{y} \in \mathcal{Y}_S$. □

Consequently, Definition 4.1 (1) includes the weakly supported nondominated points for general MOO problems, while Definitions (2) and (3) only contain the supported nondominated points (in the discrete case). Hence, in the discrete case, the weakly supported nondominated points lie on the boundary of the upper image while the supported ones lie only on the nondominated frontier, i. e., on maximally nondominated faces.

Example 4.6. *To illustrate the geometrical properties of weakly supported nondominated points, consider the outcome vectors in Figure 4.1. The outcome vectors are partitioned into two layers based on their value in the third component. The nondominated points* $y^1 = (2, 9, 1)^\top, y^2 = (3, 6, 1)^\top, y^3 = (8, 3, 1)^\top$, *and* $y^4 = (6, 5, 1)^\top$ *all share a minimum value of* $c_3 = 1$ *in their third component (pink layer), while there are other nondominated points having a value of 5 in the third component (blue layer).*

Among them, only y^1, y^2, *and* y^3 *lie on the nondominated frontier and can be obtained as optimal solutions of a weighted-sum scalarization with* $\lambda \in \Lambda_d$. *In contrast,* y^4 *does not lie on the nondominated frontier but is still on the boundary of the upper image (on a weakly nondominated face). It can be obtained as an optimal solution of a weighted-sum scalarization where some weights are zero, e.g., obtained as an optimal solution of* P_λ *with* $\lambda^\top = (0, 0, 1)^\top$, *making* y^4 *a weakly supported but not a supported nondominated point.*

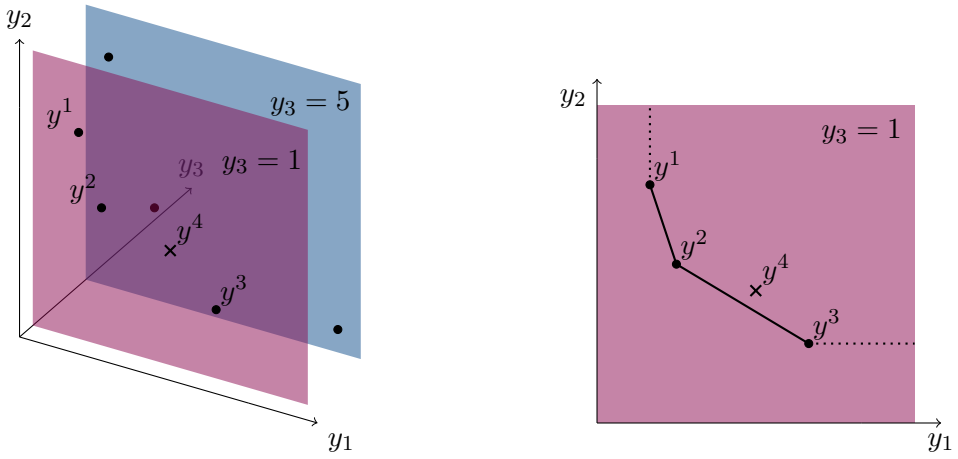


Figure 4.1: Outcome space with nondominated points $y^1, \ldots, y^4$, each with the value of 1 in the third component and the two-dimensional projection of the plane $c_3 = 1$.

Theorem 4.7. *Let* $\mathcal{Y}_S$ *and* $\mathcal{Y}_{wS}$ *be the sets of supported and weakly supported nondominated points of a MOCO problem, respectively. Then,* $\mathcal{Y}_S \subseteq \mathcal{Y}_{wS}$ *and there exist instances where* $\mathcal{Y}_S \subset \mathcal{Y}_{wS}$, *i. e., the set of supported nondominated points is a proper subset of the set of the weakly supported nondominated points.*

Proof. The inclusion $\mathcal{Y}_S \subseteq \mathcal{Y}_{wS}$ holds per definition. Consider the outcome set $\mathcal{Y} = \{y^1, y^2, y^3, y^4\}$ with $y^1 = (2, 9, 1)^\top, y^2 = (3, 6, 1)^\top, y^3 = (8, 3, 1)^\top$, and $y^4 = (6, 5, 1)^\top$, which represents the pink layer of the outcome set in Example 4.6, displayed in Figure 4.1. It is straightforward to construct an artificial MOCO problem with this outcome set.

All points y^i with $i \in \{1, \dots, 4\}$ are weakly supported nondominated points since their preimages are optimal solutions of a weighted sum problem P_λ with $\lambda = (0, 0, 1)^\top$. Since y^1,y^2 and y^3 are also supported nondominated points, since their preimages are optimal solutions of the respective weighted sum problems P_{λ^1} with $\lambda^1 = (0.7, 0.1, 0.2)^\top$, P_{λ^2} with $\lambda^2 = (0.4, 0.4, 0.2)^\top$, and P_{λ^3} with $\lambda^3 = (0.1, 0.8, 0.1)^\top$, respectively. The corresponding *(projected) weight space decomposition* is illustrated in Figure 4.2. Note that, for example, $(6, 4.2, 1)^\top \in \operatorname{conv}(\mathcal{Y}_N) \cap (y^4 - \mathbb{R}^p_{\geqq})$ implying $y^4 \notin \{y \in \operatorname{conv}(\mathcal{Y}_N) \colon \operatorname{conv}(\mathcal{Y}_N) \cap (y - \mathbb{R}^p_{\geqq}) = \{y\}\}$. By Theorem 4.5, it follows that $y_4 \notin \mathcal{Y}_S$, although $y^4 \in \mathcal{Y}_{wS}$. Thus, the set of weakly supported nondominated points is $\mathcal{Y}_{wS} = \{y^1, y^2, y^3, y^4\}$, while the set of supported nondominated points is $\mathcal{Y}_S = \{y^1, y^2, y^3\}$. Hence, in this example, the set of supported nondominated points is a proper subset of the set of weakly supported nondominated points $\mathcal{Y}_S \subset \mathcal{Y}_{wS}$. □

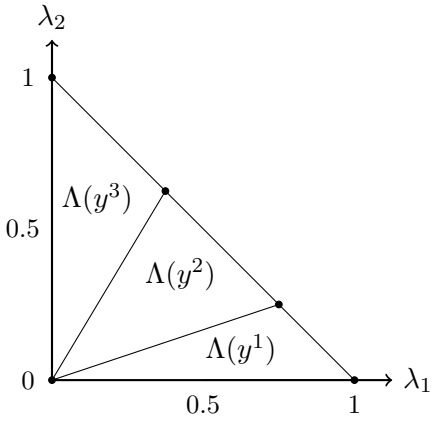


Figure 4.2: Projected weight space decomposition to the upper image of $\mathcal{Y} = \{(2, 9, 1)^\top, (3, 6, 1)^\top, (8, 3, 1)^\top, (6, 5, 1)^\top\}$ with $\lambda_3 := 1 - \lambda_2 - \lambda_1$. The set of weighting vectors associated with a point $y \in \mathcal{Y}$ is given by $\Lambda(y) := \{\lambda \in \Lambda_p^0 \colon \lambda^\top y \leq \lambda^\top y' \text{ for all } y' \in \mathcal{Y}^{\geq}\}$. For a comprehensive overview of the weight space decomposition, we refer to Przybylski et al. [2010a].

Note that in the bi-objective case $\mathcal{Y}_S = \mathcal{Y}_{wS}$ as the following lemma shows.

Lemma 4.8. *Every weakly supported nondominated point of a biobjective integer optimization problem is supported nondominated.*

Proof. Let $\bar{y} = f(\bar{x}) \in \mathbb{R}^2$ be a weakly supported but not supported nondominated point of a bi-objective integer optimization problem and $\bar{x}$ the corresponding preimage. Then there is a weighting vector $\lambda \in \Lambda_0$ such that $\bar{x}$ is an optimal solution of the weighted sum problem P_λ. Since $\bar{y}$ is not supported nondominated, one of the components of λ must be zero, w.l.o.g. let $\lambda = (1, 0)^\top$.

Since y is nondominated, there does not exist a feasible outcome vector $y = (y_1, y_2)^\top$, with $y_1 = \bar{y}_1$ and $y_2 < \bar{y}_2$. Thus, $\bar{x}$ is also an optimal solution of the weighted sum problem $P_{\lambda'}$ with $\lambda' = (1 - \varepsilon, \varepsilon)^\top$ for $\varepsilon > 0$ sufficiently small, which makes $\bar{y}$ supported nondominated. □

The clear distinction between the sets of supported nondominated and weakly supported nondominated solutions (Definition 4.3) is also necessary as the corresponding problems may differ in their output time complexity. For instance, as we will see in Section 7.3, in the case of the minimum cost flow problem, it can be shown that supported efficient solutions can be determined in *output-polynomial time*, while this is not the case for *weakly supported solutions*, unless $\mathbf{P} = \mathbf{NP}$.

Theorem 4.9. *The determination of all weakly-supported nondominated points for a MOCO problem with $p+1$ objectives is as hard as the determination of all nondominated points for a MOCO with p objectives.*

Proof. Assume an algorithm exists to determine all weakly supported nondominated points for a given MOCO problem with $p+1$ objectives. Let M_p be a MOCO with p objectives. Suppose we add an artificial objective $c^{p+1} = 0$ to our MOCO problem and denote it by M_{p+1}. In that case, we obtain a weakly efficient facet for M_{p+1}, where all nondominated points for M_p are weaklysupported nondominated points for M_{p+1}. Therefore, even the nonsupported nondominated points for M_p are weaklysupported nondominated points for M_{p+1} since they are part of the boundary of the upper image for M_{p+1}. Consequently, any algorithm that can determine all weakly supported nondominated points for M_{p+1} can determine all nondominated points for M_p. □

While the above characterization of supported nondominated points holds for MOCO problems, it does not extend to general MOO problems, and it can be shown that in the case of general MOO problems it may hold $\mathcal{Y}_S \subset \mathcal{Y}_{SNF}$. This distinction arises because the weighted sum scalarization for $\lambda \in \Lambda_p$ method determines only supported nondominated points that are also properly nondominated in the sense of Geoffrion [Geoffrion, 1968], as discussed in Ehrgott [2005]. Note that for MOCO problems every efficient solution is also properly efficient [Ehrgott, 2005]. However, for general MOO problems, the set of properly nondominated points may form a strict subset of the nondominated points on the nondominated frontier. The following example, similar to the ones in Boyd and Vandenberghe [2004] or Ehrgott [2005], illustrates this.

Example 4.10. *Consider the following MOO problem*

$$\begin{aligned} \min \quad & x_1 + 2 \\ \min \quad & x_2 + 2 \\ s.t. \quad & x_1^2 + x_2^2 \leq 1 \end{aligned}$$

Consider the points $y^1 = (1,2)^\top$ and $y^2 = (2,1)^\top$, which can be obtained with $\lambda^1 = (0,1)^\top$ and $\lambda^2 = (1,0)^\top$, respectively. It holds $y^1, y^2 \in \mathcal{Y}_{SNF}$ but y^1 and y^2 are not optimal solutions for P_λ for any $\lambda \in \Lambda$ [Boyd and Vandenberghe, 2004; Ehrgott, 2005]. Thus, $y^1, y^2 \notin \mathcal{Y}_S$. However, any point on the left lower boundary without the points $(1,2)^\top$ and $(2,1)^\top$ are both contained in $\mathcal{Y}_{SNF}$ and $\mathcal{Y}_S$. Therefore, in this example, $\mathcal{Y}_S \subset \mathcal{Y}_{SNF}$. The example is illustrated in Figure 4.3

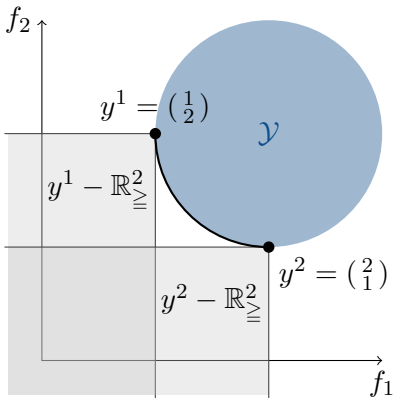


Figure 4.3: Illustration of Example 4.10 highlighting the points y^1, y^2 the points that contains the points that are both in $\mathcal{Y}_{SNF}$ and $\mathcal{Y}_{S\lambda}$ (bold curve), which together with y^1 and y^2 gives the nondominated frontier.

Theorem 4.11. *For general MOO problems it holds $\mathcal{Y}_{S\lambda} \subseteq \mathcal{Y}_{SNF}$ and there exist instances where $\mathcal{Y}_{S\lambda} \subset \mathcal{Y}_{SNF}$.*

Proof. $\mathcal{Y}_{S\lambda} \subseteq \mathcal{Y}_{SNF}$ follows directly from the proof of Theorem 4.5. Furthermore, as demonstrated in Example 4.10, there exist instances where $\mathcal{Y}_{S\lambda} \subset \mathcal{Y}_{SNF}$. □

The distinction between supported and weakly supported is sufficient for MOCO problems. However, in the context of general MOO, it may be worthwhile to introduce a finer classification, distinguishing between properly supported, supported, and weakly supported nondominated points. While this work primarily focuses on the supportedness in MOCO problems, a comprehensive analysis of supportedness in general MOO problems remains an important area for future research. Notably, recent advancements in this direction have been discussed in Chlumsky-Harttmann [2025].

4.3 Conclusion

Some previous literature use inconsistent characterizations of supported nondominated points for MOCO problems. Through counterexamples and theoretical analysis, this chapter proves that these definitions, while being equivalent in the context of MOLP, diverge in MOCO problems, yielding distinct sets of supported nondominated points with differing structural and computational properties. This emphasizes the need for precise and consistent definitions. Hence, this chapter proposes the definition of weakly supported nondominated points and establishes a clear distinction between weakly supported and supported nondominated points. This is particularly important for MOCO problems with more than two objectives.

Using these refined definitions, the following characterizations for MOCO problems hold: While the weakly supported nondominated points lie on the boundary of the upper image, the supported nondominated points lie only on the nondominated frontier, i. e., only on

maximally nondominated faces, and the nonsupported nondominated points lie in the interior of the upper image. However, these characterizations do not extend directly to general MOO problems. Hence, future research should aim to extend the characterization of supportedness beyond MOCO problems to generall MOO contexts, addressing the gaps highlighted by this chapter. An approach towards a categorization of supportedness in the context of general multi-objective optimization has been presented in [Chlumsky-Harttmann, 2025].

Nonsupported solutions may be reasonable compromise solutions and should thus not be neglected completely. Note that the difficulty in computing unsupported solutions arises in many integer and combinatorial optimization problems and is one reason for their computational complexity, in general [Ehrgott, 2000; Figueira et al., 2017]. One way to overcome this computational burden—at least to a certain degree—could be to determine unsupported solutions only in regions of the Pareto front that are not well represented by the set of supported non-dominated points.

5 Supported Nondominated Points as a Representation for Multi-Objective Integer Minimum Cost Flow Problems

While multi-objective linear optimization problems only contain supported nondominated points, the nondominated point set of multi-objective combinatorial optimization problems, such as network flow problems, also contain weakly supported and nonsupported nondominated points. These points generally outnumber the supported ones and are more challenging to determine, as they cannot be obtained as optimal solutions of weighted sum problems with weights strictly greater than zero.

This chapter considers the supported and extreme supported nondominated points as representations for the complete nondominated point set in network flow problems. Various quality metrics, such as coverage error, hypervolume ratio, and ε-indicator, are used to analyze and compare the quality of these representations. Multiple classes of network flow problems are generated to evaluate the representations. The results indicate that the supported nondominated points consistently provide high-quality representations, while considering only the extreme supported nondominated points may only yield sufficiently good representations in network flow problems where the arc capacities are quite small.

The content of this chapter is based on joint work with Lara Löhken, Michael Stiglmayr, and Kathrin Klamroth and will be submitted.

5.1 Introduction

As outlined in the chapters before, MOCO and MOIMCF belong to the class of computationally intractable problems, often containing a huge set of nondominated points. This results in high computational effort, particularly for high-dimensional problems or large instances, highlighting the need for approximation techniques and alternative methods to represent the entire nondominated point set. To address this challenge, researchers explore representation techniques that integrate quality measures (see, e.g., Sayın 2000). Several studies focus on representation methods or approximations in bi- or MOCO problems that includes approaches to assess their quality (see, e.g., Hamacher et al. 2007a; Vaz et al. 2015; Domínguez-Ríos et al. 2021; Mesquita-Cunha et al. 2023; Sayın 2024; "Generating representative sets for multiobjective discrete optimization problems with specified coverage errors" 2025). For a comprehensive overview of representations in multi-objective optimization and related methods, we refer to Faulkenberg and Wiecek [2010] and Herzel et al. [2021]

In MOIMCF problems, even the number of extreme supported nondominated points can grow exponentially with the number of vertices in the network [Ruhe, 1988]. Further-

more, Raith and Ehrgott [2009] shows that in integer bi-objective network flow problems, the number of nonsupported nondominated points is typically much larger than that of supported or extreme supported solutions, making efficient representation methods even more important. While Eusébio et al. [2014] investigates representations for BOIMCF problems, research on representations specifically for MOIMCF problems remains scarce [Hamacher et al., 2007b], underscoring the need for further studies in this area.

A recent research by Sayın [2024] demonstrated that the set of supported nondominated points can already provide high-quality representations in multi-objective discrete optimization problems, focusing specifically on binary knapsack and binary assignment problems, where the latter is a special case of the minimum cost flow problem. Beyond their representational advantages, supported nondominated points are also appealing due to their relatively straightforward computation compared to nonsupported points, which typically far outnumber them (see, e.g., Visée et al. 1998; Raith and Ehrgott 2009), and can serve as a foundation for two-phase methods (see, e.g., Pasternak and Passy 1973; Visée et al. 1998; Hamacher et al. 2007b; Przybylski et al. 2008; Przybylski et al. 2010b; Eusébio and Figueira 2009b; Dai and Charkhgard 2018).

Several studies focus on identifying or analyzing the supported or extreme supported nondominated point set across various network flow or graph problems, including bi- and multi-objective integer network flow problems [Eusébio et al., 2014; Medrano and Church, 2015; Raith and Sedeño-Noda, 2017; Könen and Stiglmayr, 2025a; Könen and Stiglmayr, 2023], bi- and multi-objective minimum spanning trees [Sourd et al., 2006; Silva and Clímaco, 2007; Correia et al., 2021], bi- and multi-objective shortest path problems [Edwin Romeijn and Smith, 1999; Sedeño-Noda and Raith, 2015; Medrano and Church, 2014], and bi- and multi-objective assignment problems [Tuyttens et al., 2000; Gandibleux et al., 2003; Przybylski et al., 2010b]. Additional work on general MOCO problems can be found in Gandibleux et al. [2001], Jesus [2015], and Sayın [2024], among others.

In the study of Sayın [2024], it is shown that $|\mathcal{Y}_S|$ and $|\mathcal{Y}_{ES}|$ are identical across all knapsack problem instances and nearly identical for all (binary) assignment problems. Consequently, the impact on the quality metrics is minimal when nonextreme supported nondominated points are added to the representation set. This implies that, for these problem classes, the extreme supported nondominated points already provide high-quality representations, and there is no need to determine further non-extreme supported nondominated points to improve the quality of the representations significantly.

However, this behavior does not extend to MOIMCF or general network problems with higher arc capacities due to the structure of such networks. As shown in Property 2.66, different solutions of the minimum cost flow problem differ only on combinations of residual cycles. Transitioning from one extreme efficient solution to another, where the corresponding extreme supported nondominated points are adjacent, might be obtained by augmenting flow along a single residual cycle. While this augmentation may affect only one cycle, the corresponding residual capacity of this cycle might be large. As a result, augmenting flow incrementally along this cycle can generate a substantial number of supported nondominated points on the face connecting these two extreme points. Additionally, other combinations of residual cycles can yield further supported nondominated points on the

same face.

Therefore, for MOIMCF problems with arc capacities greater than one, the number of supported nondominated points is expected to significantly exceed the number of extreme supported ones. The numerical experiments presented in Section 5.3 will confirm this observation.

This chapter investigates the quality of the set of (extreme) supported nondominated points as a representation of the entire nondominated set for MOIMCF problems. Various quality metrics, such as coverage error and hypervolume ratio, are used to analyze and compare the quality of these representations. Moreover, approximation indicators will be used as the ε-indicator in order to qualify the supported set as an approximation of the entire nondominated point set. The numerical results show that across various test instances, supported solutions consistently demonstrated superior representational quality, as measured by hypervolume ratio and coverage error. For all instances, the hypervolume ratio of the supported nondominated points as representations always are close to one and provides minor coverage errors. In contrast, extreme supported nondominated points yielded significantly lower quality measures, particularly as arc capacities increase.

This chapter is organized as follows. Section 5.2 gives an introduction to the mathematical definitions of the quality measures for representations and approximations. Section 5.3 describes the different instances of varying sizes and a variety of data generation schemes for the considered graph problems and reports the results of the computational experiments. The results are summarized and concluded in Section 5.4.

5.2 Representations

This section presents the most important definitions and evaluation techniques for analyzing a representation $R \subseteq \mathcal{Y}_N$ of the nondominated point set. Note that some authors allow representations to include solutions that are not efficient [Bazgan et al., 2017].

Different quality measures are used to evaluate the representations while aiming for high-quality representations. For the remainder of this study, the nondominated point set $\mathcal{Y}_N$ of the considered problem is assumed to be known. This set can be determined, for example, using the open-source implementation of the Defining Point Algorithm as described in Dächert et al. [2024].

This chapter focuses on the quality of the set of (extreme) supported nondominated points as a representation of the entire nondominated set for MOIMCF problems. These points are particularly suitable as representations due to their advantageous structure, i.e., lying on the nondominated frontier, and as demonstrated by Visée et al. [1998], for multi-objective combinatorial optimization (MOCO) problems, the number of supported solutions typically grows linearly in practice, while the size of the nondominated set grows exponentially, often outnumbering the supported nondominated points. In Section 5.3, not only are the quality measures analyzed, but also the sheer number of supported and extreme supported points compared to the number of all nondominated points.

5.2.1 Quality Metrics and Evaluation

In order to evaluate the quality of a representation $R \subseteq \mathcal{Y}_{\mathcal{N}}$, several measures are used. In addition to the measures presented in Sayın [2024] we use measures from the literature and introduce new ones. These metrics ensure that the representation accurately reflects the characteristics of the nondominated point set, such as diversity, uniformity, and coverage.

Coverage Error The first considered metric is the coverage error introduced in Sayın [2000]. The coverage error $\mathrm{CE}(R)$ is a metric used to evaluate the quality of a representation $R \subseteq \mathcal{Y}_N$, defined as the maximum distance from any nondominated point $y \in \mathcal{Y}_N$ to its closest representation point $r \in R$. The coverage error is computed as:

$$\mathrm{CE}(R) = \max_{y \in \mathcal{Y}_N} \min_{r \in R} d(y, r),$$

where $d(\cdot, \cdot)$ denotes the weighted Tchebycheff distance where the weights are inversely proportional to the criteria ranges. Therefore, for $j = 1, \ldots, p$, the weights are given by

$$w_j = \frac{1}{\max_{y \in \mathcal{Y}_N} y_j - \min_{y \in \mathcal{Y}_N} y_j}$$

and the distance is given by $d\left(y^1, y^2\right) = \max_{j=1,\ldots,p} w_j |y_j^1 - y_j^2|$.

A small coverage means that, for any nondominated point, there should be a solution in the representation whose image is sufficiently close. This metric corresponds to the one-sided Hausdorff distance between the sets R and $\mathcal{Y}_N$. Since $R \subseteq \mathcal{Y}_N$ in this study, the Hausdorff distance in the opposite direction equals zero. Consequently, $\mathrm{CE}(R)$ is precisely the Hausdorff distance between R and $\mathcal{Y}_N$, making it a topological measure.

As a pessimistic measure, $\mathrm{CE}(R)$ focuses on the worst-case scenario within the nondominated set, ensuring a robust representation quality. However, also the Median Error and Mean Error of the coverage error will be examined, as introduced in Schutze et al. [2012] and Sayın [2024]. The coverage error is computed for a particular $\bar{y} \in \mathcal{Y}_N$, as $\min_{r \in R} d(\bar{y}, r)$, determined by a representative point closest to it. The Median Error $\mathrm{ME}(R)$ for the set R is the median of these errors, while the Mean Error $\mathrm{MEAN}(R)$ is the average over all these errors. The $\mathrm{ME}(R)$ reflects that regardless of how errors are distributed across the elements of $\mathcal{Y}_N$, we can say that half of the points in the nondominated set are covered with an error of $ME(R)$ or less, and the other half are covered with an error between $\mathrm{ME}(R)$ and $\mathrm{CE}(R)$. Note that the error can be equal to zero if the nondominated point is included in the representation. Therefore, the median error can be zero if the representation contains more than half of the nondominated point set.

To evaluate the performance of a representation R in terms of its cardinality, the following measure is introduced:

$$\mathrm{OCER}(R) = \frac{\min_{R^* \subseteq \mathcal{Y}_N : |R^*| = |R|} \mathrm{CE}(R^*)}{\mathrm{CE}\,(R)}$$

The term $\min_{R^* \subseteq \mathcal{Y}_N : |R^*| = |R|} \mathrm{CE}(R^*)$ represents the representation with exactly $|R|$ points that minimizes the coverage error. Consequently, $CE(R^*)$ is the smallest achievable coverage error for any representation containing $|R|$ points. This term is closely connected to the *coverage representation problem* presented in Vaz et al. [2015], as it would be an optimal solution to this problem. The error is denoted by *optimal coverage error ratio* (OCER).

By computing $\mathrm{OCER}(R)$, we obtain a normalized error metric that indicates how close the representation coverage error $\mathrm{CE}(R)$ is to the best possible representation's coverage error with the same number of points. A value of $\mathrm{OCER}(R) = 1$ indicates that R has the best possible coverage error among all representations of size $|R|$, while values closer to zero indicate a bigger gap between $\mathrm{CE}(R)$ and $\mathrm{CE}(R^*)$. For example, a value $\mathrm{OCER}(R) = 0.5$ represents that the optimal error is only half of the actual error given by the representation R, i.e., almost twice as big as the best achievable error with this amount of points. This normalization provides an interpretable and comparable way to evaluate the quality of representations R, particularly when considering the trade-off between cardinality and representation quality. Such measures are crucial in applications where the size of the representation is constrained.

Figure 5.1 illustrates the coverage error measures of the given BOIMCF problem presented earlier in Section 2.5 in Figure 2.8, depicting the supported nondominated or extreme supported nondominated points as representations.

Hypervolume Ratio The second measure we used is the so-called *hypervolume ratio*. The hypervolume ratio for a representation R is defined as

$$\mathrm{HVR}(R) = \frac{\mathrm{HV}(R)}{\mathrm{HV}\left(\mathcal{Y}_N\right)},$$

where $\mathrm{HV}(R)$ represents the widely used hypervolume indicator for the representation R. This indicator is a well-established metric in evolutionary multi-objective optimization (see, e.g., Zitzler et al. 2003). The hypervolume indicator measures the volume of the region dominated by a set of representative points and bounded by a pre-specified reference point. Often, the Nadir point is used as the reference point. A larger hypervolume indicates a greater dominated region, reflecting a higher-quality approximation of the solution set. In our tests, we use the hypervolume ratio to quantify the quality of the representation R relative to the complete nondominated set $\mathcal{Y}_N$. A ratio closer to 1 indicates that the representation R closely approximates the entire nondominated set, providing a robust measure.

Note that the selection of the reference point can significantly influence the calculated hypervolume. For example, in a bi-objective setting, if the Nadir point is used as the reference, the two lexicographic optimal solutions will not contribute to the hypervolume. In our test instances, we use the Nadir point shifted by plus one unit in each component as the reference point. However, the influence of the lexicographical minima remains limited.

Due to the hypervolume's sensitivity to the reference point, we also consider the *range ratio* as a complementary indicator. The range ratio is independent of the reference point and evaluates the extent of the objective space covered by the representation set relative to the ideal achievable range.

Range Ratio The Range Ratio, also used in Sayın [2024], is defined as

$$\mathrm{RR}(R) = \frac{1}{p} \sum_{j=1}^{p} \frac{\max_{y \in R} y_j - \min_{y \in R} y_j}{\max_{y \in \mathcal{Y}_N} y_j - \min_{y \in \mathcal{Y}_N} y_j},$$

where p is the number of objective functions. This measure quantifies how well a representation R spans the range of each objective function compared to the full nondominated set $\mathcal{Y}_N$.

A value of $\mathrm{RR}(R) = 1$ indicates that the representation R covers the full range of the objective values for each objective function, matching the extent of the entire nondominated set. Conversely, values closer to zero imply that the representation fails to adequately cover the range of objective values, leaving gaps in its coverage at the problem's boundaries. The Range Ratio is particularly useful for assessing the ability of a representation to reach the extremes of the nondominated set. This is important in multi-objective optimization, where capturing the boundary values often provides valuable insights into the trade-offs between competing objectives. Moreover, some solution algorithms fail to reach these extremal solutions. By averaging across all objective functions, the Range Ratio provides a comprehensive view of the representation's ability to reflect the problem's full scope.

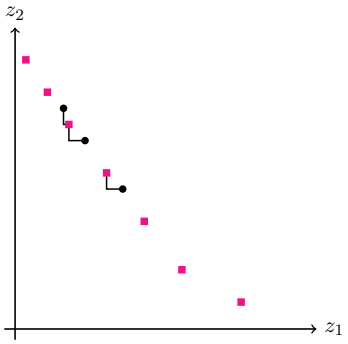
However, since this work considers the supported nondominated points as the representation set, which includes all extreme supported nondominated points, the Range Ratio is expected to be close to 1. Notably, for bi-objective problems, the Range Ratio will always equal 1, as the representation (with extreme supported nondominated points) fully spans the range of each objective function. Figure 5.2 illustrates the hypervolume and range ratios of the given BOIMCF problem of Figure 2.8, depicting the supported nondominated or extreme supported nondominated points as representations.

ε-indicator The last metric presented is the ε-indicator, a widely recognized performance measure in multi-objective optimization, particularly for heuristic solution methods and approximation algorithms [Papadimitriou and Yannakakis, 1991]. This indicator evaluates how well a representation set R approximates the target set $\mathcal{Y}_N$ by determining the smallest factor ε by which the elements of $\mathcal{Y}_N$ must be scaled so that for each $y \in \mathcal{Y}_N$ there exists a $r \in R$ which weakly dominates $\mathcal{Y}_N$. The ε-indicator is calculated following the definitions provided in Zitzler et al. [2003] and Vaz et al. [2015] as:

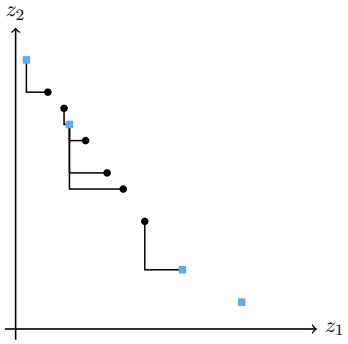
$$I_\varepsilon(R) = \max_{y \in \mathcal{Y}_N} \min_{r \in R} \varepsilon(r, y),$$

where

$$\varepsilon(r, y) = \max_{i \in (1,\dots,p]} \frac{r^i}{y^i}$$



$\mathrm{CE}(\mathcal{Y}_{SN}) = 0.075$
$\mathrm{MECE}(\mathcal{Y}_{SN}) = 0$
$\mathrm{MeanCE}(\mathcal{Y}_{SN}) = 0.0216$
$\mathrm{OCVR}(\mathcal{Y}_{SN}) = 1$



$\mathrm{CE}(\mathcal{Y}_{ES}) = 0.267$
$\mathrm{MECE}(\mathcal{Y}_{ES}) = 0.075$
$\mathrm{MeanCE}(\mathcal{Y}_{ES}) = 0.0941$
$\mathrm{OCVR}(\mathcal{Y}_{ES}) = 0.749$

Figure 5.1: Illustration of the coverage error for the nondominated point set of the bi-objective integer minimum cost flow (BOIMCF) problem from Figure 2.8. In the left figure, the supported nondominated points are used as representation, while the right figure depicts the extreme supported nondominated points. Among all nondominated points, the representative points are highlighted as colored rectangles.

This formulation guarantees that for every $y \in \mathcal{Y}_N$, there exists an $r \in R$ such that r weakly dominates y. The ε-indicator provides an intuitive measure of approximation quality. If $R = \mathcal{Y}_N$ it holds $I_\varepsilon(R) = 1$. Also note that the factor I_ε defined above corresponds to the approximation ratio $(1 + \varepsilon)$ typically used in the context of approximation schemes; see, for example, Papadimitriou and Yannakakis [1991].

This concept is closely related to the coverage representation problem; achieving good coverage often corresponds to obtaining a good ε-indicator, and vice versa, as demonstrated in Vaz et al. [2015].

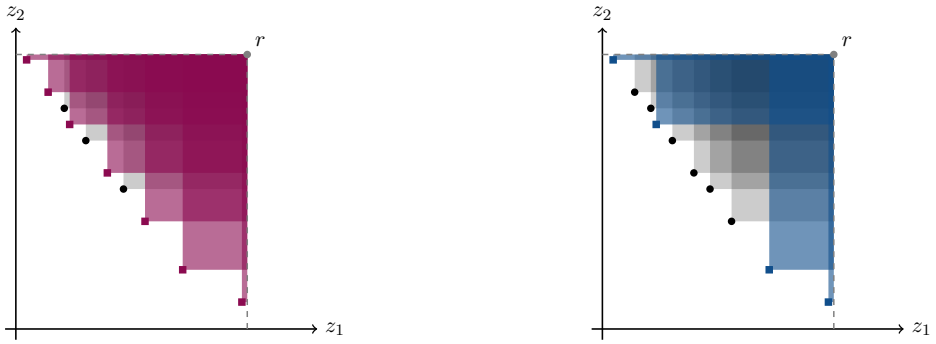
To further assess the quality of a representation R with respect to its cardinality and to normalize the result to a value between 0 and 1, we introduce the *optimal ε-indicator ratio* as

$$\mathrm{O}\, I_\varepsilon(R) = \frac{1}{I_\varepsilon(R)} \min_{R^* \subseteq \mathcal{Y}_N : |R^*| = |R|} I_\varepsilon(R^*).$$

For the problem shown in Figure 2.8 it holds $I_\varepsilon(\mathcal{Y}_S) = 1.025$ and $I_\varepsilon(\mathcal{Y}_{ES}) \approx 1.0965$, considering $\mathcal{Y}_S$ and $\mathcal{Y}_{ES}$ as respective representations.

5.3 Numerical Experiments

This section presents the implementation details and provides a numerical evaluation of the quality of the two representation sets, $\mathcal{Y}_S$ and $\mathcal{Y}_{ES}$. Their effectiveness as representations is evaluated across a diverse set of test instances. The section aims to report and



$\text{HVR}(\mathcal{Y}_{SN}) = 0.97$
$\text{RR}(\mathcal{Y}_{SN}) = 1$

$\text{HVR}(\mathcal{Y}_{ES}) = 0.73$
$\text{RR}(\mathcal{Y}_{ES}) = 1$

Figure 5.2: Illustration of the hypervolume ratio for the nondominated point set of the bi-objective integer minimum cost flow (BOIMCF) problem from Figure 2.8. In the left figure the supported nondominated points are used as representation, while the right figure depicts the extreme supported nondominated points. Among all nondominated points, the representative points are highlighted as colored rectangles. The reference point r is chosen as the Nadir point with plus one in each component.

compare the results using several quality metrics, providing a comprehensive analysis of the quality of the supported nondominated and extreme supported nondominated points as representations under different schemes for MOIMCF problems.

The general setup of the computational experiments is as follows. For each problem instance, the complete nondominated point set is computed using the open-source implementation of the Defining Point Algorithm from Dächert et al. [2024]. The subsets of supported and extreme supported nondominated points are extracted from those nondominated sets by solving the subproblems introduced in Sayın [2024], as detailed in Section 5.3.1. These subproblems are solved using a C++ code that uses a CPLEX Callable Library.

The resulting sets are then evaluated using the set of metrics presented in the previous section. Metric evaluations related to coverage and the ε-indicator are implemented in C++. For hypervolume computations, we use the instance-specific Nadir point shifted by one unit in each coordinate as the reference point. The hypervolume values are computed using the open-source C implementation (Version 1.3) described in Fonseca et al. [2006].

All computations are conducted on a computer with an Intel® Core™i8-8700U CPU 3.20 GHz processor with 32 GB RAM, using a LINUX operating system. As this is primarily an empirical study focused on representation quality, we do not report runtime performance. However, we note that computing the full nondominated set becomes computationally intensive even for moderately sized instances. In particular, we encountered scalability limitations for large instances or instances with more than five objectives, where generating the entire nondominated set was too computationally demanding.

The remainder of this section is structured as follows. This section describes how to extract the supported and extreme supported nondominated points from the full set of nondominated points. Then, it presents the generation process of different test instance classes, followed by a detailed analysis and discussion of the numerical results.

5.3.1 Identifying Supported and Extreme Supported Nondominated Points

Given the complete list of nondominated points, the two linear programs introduced in Sayın [2024] are used to identify the sets of all supported and extreme supported nondominated points. These LP formulations serve as subproblems for classification and rely on input from the entire nondominated set or its supported subset.

Identifying Supported Nondominated Points To identify whether a nondominated point $y^k \in \mathcal{Y}_N$ is supported, i.e., $y^k \in \mathcal{Y}_S$, the linear program, denoted by S_{y^k}, is solved. Importantly, this check requires the complete nondominated set $\mathcal{Y}_N$ as input.

$$
\begin{aligned}
\min \quad & z^k = \max \sum_{i=1}^{p} \lambda_i & & \\
\text{s.t.} \quad & \sum_{i=1}^{p} \lambda_i y_i^k \leq \sum_{i=1}^{p} \lambda_i y_i^j & & j = 1, \ldots, |\mathcal{Y}_N| \\
& \sum_{i=1}^{p} \lambda_i = 1 & & \\
& \lambda_i \geq 0 & & i = 1, \ldots, p
\end{aligned}
\tag{S_{y^k}}
$$

This linear program contains p non-negative variables, representing the weights in a weighted sum subproblem (P_λ). The first set of constraints ensures that y^k is at least as good as any other nondominated point under the weighted objective. The constraint $\sum_{i=1}^{p} \lambda_i = 1$ guarantees that the weight vector is normalized. Consequently, this formulation checks whether a weight vector exists for which y^k corresponds to the image of an optimal solution to (P_λ). In this case, y^k is a supported nondominated point. The result of this test is revealed by the optimal objective value, as captured in the following theorem.

Theorem 5.1 (Sayın 2024)**.** *A nondominated point y^k is supported, i.e., $y^k \in \mathcal{Y}_S$ if and only if $z^k = 1$ in S_{y^k}.*

Identifying Extreme Supported Nondominated Points To determine whether a supported nondominated point $y^k \in \mathcal{Y}_{ES}$, the following linear program, denoted by E_{y^k}, is solved. This check requires the complete set of supported nondominated points as input.

$$
\begin{aligned}
\min \quad & z^k = \min \alpha_k \\
\text{s.t.} \quad & \sum_{j=1}^{|\mathcal{Y}_S|} \alpha_j y_i^j = y_i^k \qquad i = 1, \dots, p \\
& \sum_{j=1}^{|\mathcal{Y}_S|} \alpha_j = 1 \\
& \alpha_j \geq 0 \qquad j = 1, \dots, |\mathcal{Y}_S|
\end{aligned}
\tag{E_{y^k}}
$$

This formulation is a linear program with $\mathcal{Y}_S$ non-negative variables. The first constraint represents the nondominated point y^k as a linear combination of all nondominated points, while $\sum_{j=1}^{|\mathcal{Y}_S|} \alpha_j = 1$, together with the non-negativity constraints, ensures that y^k is expressed as a convex combination of certain nondominated points. Again, the classification result is determined by the optimal objective value.

Theorem 5.2 (Sayın 2024)**.** *A nondominated point $y^k \in \mathcal{Y}_N$ is extreme supported, i.e., $y^k \in \mathcal{Y}_{ES}$ if and only if $z^k = 1$ in E_{y^k}.*

5.3.2 Test Instances

The first set of computational experiments summarized in Table 5.1 evaluates MOIMCF problem classes generated using the NETGEN network generator [Klingman et al., 1974]. In total, 11 problem classes were considered, with each class comprising 15 randomly generated instances.

Each NETGEN instance is defined by several parameters, including the number of nodes and arcs, the number of supply and sink nodes, as well as upper bounds for arc costs, arc capacities, and the total network supply. These 11 problem classes differ in the number of nodes and arcs, ranging from 20 to 2000 nodes and 40 to 4000 arcs, to explore their influence on the structure of the nondominated set. Given their strong impact on the number of supported nondominated points and efficient solutions, these two parameters were treated as independent variables. All other NETGEN parameters were held constant across the instances to isolate the effects of the network size. Specifically, each instance was configured with two nodes acting as supply nodes and two as sink nodes, a maximum arc cost of 10 for all three objectives, a maximum upper capacity of 50, and a total supply of 50. Note that we ensured to generate no instances with a feasible ideal point.

The results for these NETGEN-generated instances are presented in Table 5.1. For each problem class, the table displays the number of extreme supported nondominated points, supported nondominated points, supported efficient solutions, their ratio, and the hypervolume indicator as well as the range ratio for the supported nondominated and extreme supported nondominated points as representations. In addition, Table 5.2 reports the coverage error and ε-indicator for the same instances. We display the minimum, maximum, and mean for each class across the 15 instances. An overall summary of these aggregated metrics across all 11 classes is provided in Table 5.3.

We fixed the number of nodes $n = 50$ for the second set of test instances while keeping all other NETGEN parameters consistent with those used in the first experiment. The only parameter varied was the number of arcs, allowing a systematic evaluation of how the density, i.e., the arc-to-node ratio influences the structure and quality of the representation sets, as well as the corresponding ratios of $|\mathcal{Y}_S|/|\mathcal{Y}_N|$ and $|\mathcal{Y}_{ES}|/|\mathcal{Y}_N|$. The results are summarized in Table 5.5, and the trends are visualized in Figure 5.3.

The third set of instance classes consists of networks with a fixed number of nodes $n = 50$ and $m = 200$ arcs, while all other parameters remain identical to those used in the first experiment. In this experiment, the only parameter varied is the maximum arc capacity, denoted $c^{\max}$, which takes values in $\{1, 5, 50, 100\}$. Notably, when $c^{\max} = 1$, the problem becomes a binary problem. The results for these instances are summarized in Table 5.6, with the corresponding trends illustrated in Figure 5.4.

The fourth set of instance classes investigates the impact of dimensionality by varying the number of objectives from 3 to 5. Here, we use smaller networks with $n = 10$ nodes and $m = 20$ arcs, keeping all other parameters fixed. Due to the increased computational complexity with higher dimensions, we encountered performance limitations and were unable to extend this analysis to larger instances or problems with more than five objectives. Table 5.7 summarizes the results.

In the fifth set of instances classes, we study the effect of cost correlation between objectives. We consider bi-objective problems and adopt the cost construction approach introduced in Sayın [2024], maintaining all other NETGEN parameters constant. We consider different classes with varying numbers of nodes but the same arc-to-node ratio. The number of nodes takes values in $[20, 50, 100, 200, 400]$, and the number of arcs equals $m = 4n$. For correlation levels $\ell \in \{1, 2, 3\}$, we generate a core cost matrix $T = (t_{ij})$, where each t_{ij} is randomly drawn from the interval $[1, 5\ell]$. For each arc $(i, j) \in A$, additional noise terms $\hat{c}^k_{ij}$ are sampled uniformly from $[1, 20 - 5\ell]$, and the final cost components are defined as $c^k_{ij} = t_{ij} + \hat{c}^k_{ij}$ for each objective $k = 1, 2$. In the case of no correlation, cost components for both objectives are generated independently and uniformly from the range $[1, 20]$. Note that we do not guarantee non-feasibility of the ideal point in these instances. In fact, for higher correlation levels, it was frequently observed that the ideal point was feasible, and therefore, in these instances, it holds that $|\mathcal{Y}_N| = 1$. Table 5.8 shows the mean values over all instances for each class and each correlation level.

For all bi-objective instances with no correlation level ($\ell = 0$), we compute $\mathrm{OCER}(\mathcal{Y}_S)$ and $\mathrm{OCER}(\mathcal{Y}_{ES})$ to assess the quality of $\mathcal{Y}_S$ and $\mathcal{Y}_{ES}$ as representations in terms of their cardinality. All results are presented in Table 5.9. These computations were implemented in Python using the dynamic programming approach of Vaz et al., 2015. Note that the weighted Tchebycheff distance is used once again. We do not present the values for $\mathrm{O}\, I_\varepsilon(R)$ as they do not provide new insights compared to $\mathrm{OCER}(R)$.

The final test classes consist of Netgen instances constructed using parameters based on the test sets N01, N04, N07, and N12 from Raith and Ehrgott, 2009. These instances feature a greater number of supply, source nodes, and sink nodes compared to those considered previously. Due to the higher supply and fewer transshipment nodes with zero demand/supply, the ratio of $\mathcal{Y}_S$ to $\mathcal{Y}_N$ is expected to be lower than in the earlier test sets.

The following parameters are fixed across all problem instances: `mincost` = 0, `maxcost` = 100, `capacitated` = 100, `mincap` = 0, and `maxcap` = 50. The varying parameters are summarized in Table 5.10. Each class again includes 15 instances per parameter setting. Table 5.11 and Table 5.12 present the results for the bi-objective and tri-objective cases, respectively. Notably, for $p = 3$, we were only able to generate results for class N01. For classes N04, N07, and N12, the number of nondominated points increased drastically, making the computation too demanding. For instance, class N04 consists of an average of 37,100.53 nondominated points across the 15 instances, illustrating the rapid growth in solution complexity even with only three objectives.

The following section presents a detailed discussion of the role and influence of each parameter setting across the different instance classes.

Table 5.1: Numerical results for the different instance classes generated with NETGEN.

Class $p = 3$		$\lvert\mathcal{Y}_N\rvert$	$\lvert\mathcal{Y}_S\rvert$	$\lvert\mathcal{Y}_{ES}\rvert$	$\frac{\lvert\mathcal{Y}_S\rvert}{\lvert\mathcal{Y}_N\rvert}$	$\frac{\lvert\mathcal{Y}_{ES}\rvert}{\lvert\mathcal{Y}_N\rvert}$	HVR($\mathcal{Y}_S$)	HVR($\mathcal{Y}_{ES}$)	RR($\mathcal{Y}_S$)	RR($\mathcal{Y}_{ES}$)
1	min	3.0	3.0	2.0	0.234	0.006	0.972	0.213	0.954	0.954
$n = 20$	max	3269.0	1352.0	19.0	1.0	0.667	1.0	0.881	1.0	1.0
$m = 40$	mean	361.0	183.0	6.6	0.804	0.156	0.997	0.645	0.994	0.994
2	min	38.0	13.0	5.0	0.195	0.006	0.986	0.591	0.97	0.97
$m = 20$	max	6952.0	1359.0	77.0	1.0	0.158	1.0	0.944	1.0	1.0
$m = 80$	mean	1995.067	579.2	26.8	0.5	0.036	0.996	0.85	0.997	0.997
3	min	2.0	2.0	2.0	0.242	0.018	0.918	0.065	0.933	0.933
$n = 50$	max	2037.0	492.0	36.0	1.0	1.0	1.0	1.0	1.0	1.0
$m = 100$	mean	489.4	197.333	14.867	0.627	0.143	0.992	0.703	0.992	0.992
4	min	69.0	52.0	7.0	0.109	0.005	0.979	0.575	0.991	0.991
$n = 50$	max	21248.0	2317.0	115.0	1.0	0.145	1.0	0.953	1.0	1.0
$m = 200$	mean	2599.867	531.2	35.467	0.444	0.04	0.995	0.87	0.998	0.998
5	min	3.0	3.0	2.0	0.229	0.02	0.977	0.634	0.941	0.941
$n = 100$	max	2710.0	621.0	64.0	1.0	0.667	1.0	0.942	1.0	1.0
$m = 200$	mean	600.333	218.267	21.2	0.541	0.102	0.994	0.843	0.993	0.993
6	min	150.0	84.0	14.0	0.159	0.006	0.991	0.79	0.972	0.972
$n = 100$	max	15317.0	2497.0	225.0	0.886	0.2	0.999	0.971	1.0	1.0
$m = 400$	mean	4123.933	886.533	60.067	0.377	0.035	0.997	0.912	0.996	0.996
7	min	30.0	14.0	9.0	0.111	0.006	0.978	0.774	0.943	0.943
$n = 200$	max	7262.0	1447.0	72.0	0.672	0.41	0.999	0.982	1.0	1.0
$m = 400$	mean	2418.733	468.6	32.267	0.337	0.092	0.992	0.89	0.993	0.993
8	min	41.0	37.0	8.0	0.085	0.006	0.99	0.632	0.974	0.974
$n = 200$	max	21579.0	1840.0	174.0	0.902	0.244	1.0	0.97	1.0	1.0
$m = 800$	mean	5851.467	942.333	77.133	0.264	0.035	0.996	0.905	0.998	0.998
9	min	422.0	151.0	23.0	0.102	0.01	0.984	0.794	0.973	0.973
$n = 1000$	max	4252.0	801.0	106.0	0.391	0.081	0.998	0.983	1.0	1.0
$m = 2000$	mean	1912.933	403.733	54.267	0.247	0.035	0.994	0.925	0.996	0.996
10	min	176.0	107.0	17.0	0.11	0.003	0.993	0.76	0.956	0.956
$n = 1000$	max	25648.0	3021.0	312.0	0.608	0.125	0.999	0.98	1.0	1.0
$m = 4000$	mean	8484.333	1384.467	128.2	0.204	0.024	0.996	0.939	0.995	0.995
11	min	49.0	49.0	3.0	0.078	0.007	0.985	0.397	0.968	0.968
$n = 2000$	max	10348.0	979.0	140.0	1.0	0.077	1.0	0.976	1.0	1.0
$m = 4000$	mean	2646.8	428.2	58.6	0.393	0.039	0.995	0.867	0.996	0.996

5.3.3 Results

In the study of Sayın [2024], it is shown that $\mathcal{Y}_S$ and $\mathcal{Y}_{ES}$ yield high-quality representations for the binary knapsack and the binary assignment problem. However, in these problem classes, the sizes of the sets $|\mathcal{Y}_S|$ and $|\mathcal{Y}_{ES}|$ were identical across all knapsack instances

Table 5.2: Numerical results for the different instance classes generated with NETGEN.

Class $p = 3$		CE($\mathcal{Y}_{\mathrm{S}}$)	CE($\mathcal{Y}_{\mathrm{ES}}$)	ME($\mathcal{Y}_{\mathrm{S}}$)	ME($\mathcal{Y}_{\mathrm{ES}}$)	MEAN($\mathcal{Y}_{\mathrm{S}}$)	MEAN($\mathcal{Y}_{\mathrm{ES}}$)	$I_\varepsilon(\mathcal{Y}_S)$	$I_\varepsilon(\mathcal{Y}_{ES})$
1	min	0.0	0.289	0.0	0.0	0.0	0.098	1.0	1.009
$n = 20$	max	0.187	0.5	0.024	0.232	0.024	0.221	1.005	1.111
$m = 40$	mean	0.059	0.396	0.003	0.149	0.007	0.165	1.002	1.046
2	min	0.0	0.161	0.0	0.04	0.0	0.049	1.0	1.021
$n = 20$	max	0.107	0.464	0.074	0.229	0.059	0.242	1.009	1.085
$m = 80$	mean	0.037	0.289	0.011	0.104	0.01	0.109	1.004	1.055
3	min	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0
$n = 50$	max	0.4	0.8	0.2	0.4	0.175	0.374	1.006	1.04
$m = 100$	mean	0.093	0.323	0.019	0.137	0.022	0.138	1.002	1.026
4	min	0.0	0.138	0.0	0.047	0.0	0.048	1.0	1.022
$n = 50$	max	0.146	0.361	0.031	0.136	0.033	0.143	1.01	1.084
$m = 200$	mean	0.058	0.247	0.009	0.084	0.01	0.091	1.005	1.055
5	min	0.0	0.15	0.0	0.0	0.0	0.039	1.0	1.002
$n = 100$	max	0.36	0.5	0.051	0.141	0.043	0.167	1.005	1.081
$m = 200$	mean	0.101	0.293	0.012	0.084	0.014	0.107	1.003	1.026
6	min	0.027	0.12	0.0	0.033	0.002	0.038	1.003	1.011
$n = 100$	max	0.164	0.423	0.015	0.096	0.023	0.102	1.01	1.113
$m = 400$	mean	0.078	0.242	0.006	0.061	0.009	0.07	1.006	1.056
7	min	0.031	0.141	0.0	0.029	0.007	0.053	1.001	1.002
$n = 200$	max	0.345	0.439	0.071	0.175	0.073	0.125	1.005	1.072
$m = 400$	mean	0.094	0.27	0.014	0.083	0.02	0.088	1.003	1.025
8	min	0.019	0.094	0.0	0.03	0.002	0.033	1.002	1.014
$n = 200$	max	0.127	0.437	0.026	0.077	0.023	0.122	1.01	1.103
$m = 800$	mean	0.064	0.204	0.009	0.05	0.009	0.06	1.006	1.048
9	min	0.042	0.098	0.006	0.026	0.007	0.029	1.002	1.006
$n = 1000$	max	0.173	0.418	0.027	0.204	0.026	0.201	1.004	1.038
$m = 2000$	mean	0.101	0.204	0.014	0.061	0.014	0.068	1.003	1.018
10	min	0.019	0.077	0.0	0.017	0.006	0.021	1.004	1.022
$n = 1000$	max	0.133	0.388	0.015	0.143	0.02	0.138	1.012	1.249
$m = 4000$	mean	0.062	0.187	0.008	0.05	0.009	0.056	1.006	1.051
11	min	0.0	0.105	0.0	0.028	0.0	0.034	1.0	1.006
$n = 2000$	max	0.208	0.483	0.026	0.217	0.03	0.231	1.008	1.316
$m = 8000$	mean	0.067	0.257	0.009	0.083	0.01	0.093	1.003	1.06

and nearly identical across all binary assignment instances. Consequently, the impact of non-extreme supported nondominated points on quality metrics was minimal. This indicates that, for these specific problem classes, the extreme supported nondominated points alone already constitute a high-quality representation, and there is little to no benefit in identifying additional non-extreme supported points to improve representation quality.

This behaviour, however, does not extend to the MOIMCF or general network problems with higher arc capacities due to the structure of such networks. As seen in Schrijver, 2003; Könen et al., 2022b, different solutions of the minimum cost flow problem differ only on combinations of residual cycles (or a linear number of induced cycles). Transitioning from one extreme efficient solution to another, where the corresponding extreme supported nondominated points are adjacent, might be obtained by augmenting flow along a single residual cycle. Although such augmentation may involve only one cycle, the residual capacity of that cycle can be large. Incrementally augmenting the flow along this cycle can therefore generate a substantial number of supported nondominated points on the face connecting these two extreme points. Moreover, other combinations of residual cycles can yield additional supported nondominated points on the same face. The results of all test

Table 5.3: Overall mean values across all instance classes.

Metric	Mean
$\frac{\lvert\mathcal{Y}_S\rvert}{\lvert\mathcal{Y}_N\rvert}$	0.431
$\frac{\lvert\mathcal{Y}_{ES}\rvert}{\lvert\mathcal{Y}_N\rvert}$	0.067
$\mathrm{HVR}(\mathcal{Y}_S)$	0.995
$\mathrm{HVR}(\mathcal{Y}_{ES})$	0.858
$\mathrm{CE}(\mathcal{Y}_S)$	0.074
$\mathrm{CE}(\mathcal{Y}_{ES})$	0.265
$I_\varepsilon(\mathcal{Y}_S)$	1.004
$I_\varepsilon(\mathcal{Y}_{ES})$	1.042

Table 5.4: Inverted ratio of selected metrics.

Metric Ratio	Value
$\frac{\lvert\mathcal{Y}_S\rvert}{\lvert\mathcal{Y}_N\rvert} / \frac{\lvert\mathcal{Y}_{ES}\rvert}{\lvert\mathcal{Y}_N\rvert}$	6.43
$\frac{\mathrm{HVR}(\mathcal{Y}_S)}{\mathrm{HVR}(\mathcal{Y}_{ES})}$	1.16
$\frac{\mathrm{CE}(\mathcal{Y}_S)}{\mathrm{CE}(\mathcal{Y}_{ES})}$	0.28
$\frac{I_\varepsilon(\mathcal{Y}_S)}{I_\varepsilon(\mathcal{Y}_{ES})}$	0.96

Table 5.5: Numerical results for class $n = 50$ but differing numbers of arcs. All classes again contain 15 instances.

Class $p = 3$		$\lvert\mathcal{Y}_N\rvert$	$\lvert\mathcal{Y}_S\rvert$	$\lvert\mathcal{Y}_{ES}\rvert$	$\frac{\lvert\mathcal{Y}_S\rvert}{\lvert\mathcal{Y}_N\rvert}$	$\frac{\lvert\mathcal{Y}_{ES}\rvert}{\lvert\mathcal{Y}_N\rvert}$	$\mathrm{HVR}(\mathcal{Y}_S)$	$\mathrm{HVR}(\mathcal{Y}_{ES})$	$\mathrm{CE}(\mathcal{Y}_S)$	$\mathrm{CE}(\mathcal{Y}_{ES})$
$m = 100$	min	2.0	2.0	2.0	0.242	0.018	0.918	0.065	0.0	0.0
	max	2037.0	492.0	36.0	1.0	1.0	1.0	1.0	0.4	0.8
	mean	489.4	197.333	14.867	0.627	0.143	0.992	0.703	0.093	0.323
$m = 200$	min	69.0	52.0	7.0	0.109	0.005	0.979	0.575	0.0	0.138
	max	21248.0	2317.0	115.0	1.0	0.145	1.0	0.953	0.146	0.361
	mean	2599.867	531.2	35.467	0.444	0.04	0.995	0.87	0.058	0.247
$m = 300$	min	319.0	198.0	14.0	0.131	0.005	0.992	0.56	0.018	0.139
	max	18304.0	2745.0	147.0	0.715	0.125	0.999	0.967	0.105	0.382
	mean	6282.2	1424.8	64.667	0.326	0.019	0.997	0.893	0.043	0.232
$m = 400$	min	289.0	142.0	8.0	0.124	0.004	0.994	0.76	0.017	0.113
	max	20014.0	9593.0	230.0	0.732	0.048	1.0	0.975	0.064	0.447
	mean	7584.867	2074.733	74.4	0.347	0.016	0.997	0.909	0.042	0.202
$m = 600$	min	995.0	442.0	22.0	0.178	0.005	0.996	0.893	0.016	0.115
	max	19155.0	4670.0	159.0	0.616	0.036	0.999	0.968	0.071	0.304
	mean	8653.6	2294.267	80.467	0.31	0.013	0.998	0.93	0.036	0.201

instances discussed in the previous section form the foundation for this argumentation.

Based on the results from the first set of problem classes, presented in Tables 5.1 and 5.2 and summarized in Tables 5.3 and 5.4, several key observations can be made. In this instances, we observe a substantial difference between $\mathcal{Y}_S$ and $\mathcal{Y}_{ES}$, with $|\mathcal{Y}_S|$ being significantly larger than $|\mathcal{Y}_{ES}|$. This outcome aligns with expectations, given the high maximum arc capacity of 50. On average across all instance classes, the ratio $|\mathcal{Y}_S|/|\mathcal{Y}_N|$ was 43.1%, while $|\mathcal{Y}_{ES}|/|\mathcal{Y}_N|$ was only 6.7%, making $\mathcal{Y}_S$ approximately 6.43 times larger than $\mathcal{Y}_{ES}$ in terms of cardinality.

This disparity is reflected in the evaluation metrics, which demonstrate the superior representational quality of $\mathcal{Y}_S$. Across all test instances, the mean hypervolume ratio $\mathrm{HVR}(\mathcal{Y}_S)$ reached 99.5%, indicating that $\mathcal{Y}_S$ captures nearly the entire achievable hypervolume. In contrast, $\mathrm{HVR}(\mathcal{Y}_{ES})$ averaged 85.5%, a significantly lower value. On average, across 11 three-dimensional test classes, each consisting of 15 instances, the hypervolume

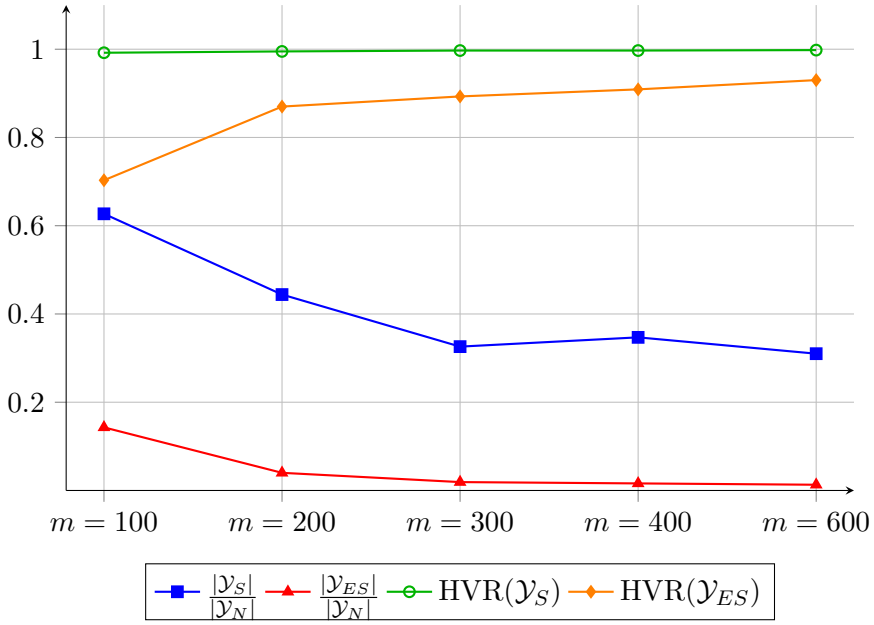


Figure 5.3: Mean values for instances with varying arc numbers ($n = 50$, $p = 3$).

ratio of supported nondominated points was about 16% higher than that of the extreme supported points. Note that the Range Ratio was identical for both sets and almost equal to one in each instance. Hence, both sets can cover the full range of the objective values.

Additionally, this performance advantage is mirrored in the coverage error metric. The mean coverage error for $\mathcal{Y}_S$ was only 28% of that for $\mathcal{Y}_{ES}$, further underscoring its superior quality. Specifically, the coverage error $\mathrm{CE}(\mathcal{Y}_S)$ over all instance classes was just 0.074, indicating that the coverage error given by the supported nondominated points as representation was quite small.

Due to the high ratio of supported nondominated points in the overall set, the mean of

Table 5.6: Numerical results for class 2 but with differing numbers of maximal capacity. All classes again contain 15 instances.

Class $p = 3$		$\lvert\mathcal{Y}_N\rvert$	$\lvert\mathcal{Y}_S\rvert$	$\lvert\mathcal{Y}_{ES}\rvert$	$\frac{\lvert\mathcal{Y}_S\rvert}{\lvert\mathcal{Y}_N\rvert}$	$\frac{\lvert\mathcal{Y}_{ES}\rvert}{\lvert\mathcal{Y}_N\rvert}$	$\mathrm{HVR}(\mathcal{Y}_S)$	$\mathrm{HVR}(\mathcal{Y}_{ES})$	$\mathrm{CE}(\mathcal{Y}_S)$	$\mathrm{CE}(\mathcal{Y}_{ES})$
$c^{\max} = 1$	min	3.0	3.0	3.0	0.464	0.464	0.879	0.879	0.0	0.0
	max	28.0	14.0	14.0	1.0	1.0	1.0	1.0	0.565	0.565
	mean	11.8	7.4	7.333	0.722	0.711	0.977	0.972	0.237	0.256
$c^{\max} = 5$	min	55.0	27.0	25.0	0.188	0.121	0.98	0.947	0.089	0.106
	max	882.0	166.0	129.0	0.519	0.455	0.997	0.995	0.218	0.277
	mean	308.333	82.0	59.267	0.352	0.264	0.987	0.971	0.14	0.182
$c^{\max} = 50$	min	69.0	52.0	7.0	0.109	0.005	0.979	0.575	0.0	0.138
	max	21248.0	2317.0	115.0	1.0	0.145	1.0	0.953	0.146	0.361
	mean	2599.867	531.2	35.467	0.444	0.04	0.995	0.87	0.058	0.247
$c^{\max} = 100$	min	23.0	12.0	3.0	0.099	0.003	0.994	0.626	0.019	0.186
	max	30454.0	3027.0	81.0	0.949	0.13	1.0	0.956	0.177	0.468
	mean	4880.467	840.733	30.533	0.442	0.027	0.997	0.842	0.056	0.308

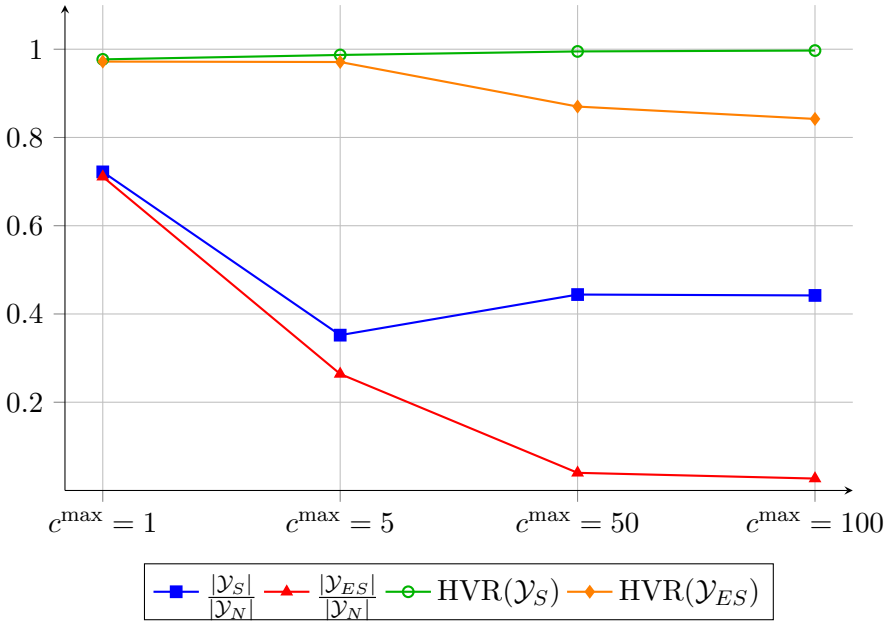


Figure 5.4: Mean values for the instances with varying maximal capacity ($p = 3$).

Table 5.7: Numerical results for ($n = 10, m = 20$) with different number of objectives. The classes again contain 15 instances.

		$\lvert\mathcal{Y}_N\rvert$	$\lvert\mathcal{Y}_S\rvert$	$\lvert\mathcal{Y}_{ES}\rvert$	$\frac{\lvert\mathcal{Y}_S\rvert}{\lvert\mathcal{Y}_N\rvert}$	$\frac{\lvert\mathcal{Y}_{ES}\rvert}{\lvert\mathcal{Y}_N\rvert}$	$\mathrm{HVR}(\mathcal{Y}_S)$	$\mathrm{HVR}(\mathcal{Y}_{ES})$	$\mathrm{CE}(\mathcal{Y}_S)$	$\mathrm{CE}(\mathcal{Y}_{ES})$
$p = 3$	min	2.0	2.0	2.0	0.379	0.016	0.945	0.015	0.0	0.0
	max	1023.0	388.0	16.0	1.0	1.0	1.0	1.0	0.133	0.5
	mean	126.867	76.6	4.933	0.833	0.2	0.994	0.575	0.041	0.371
$p = 4$	min	2.0	2.0	2.0	0.469	0.008	0.995	0.002	0.0	0.0
	max	2743.0	1286.0	23.0	1.0	1.0	1.0	1.0	0.316	0.5
	mean	296.267	178.933	6.867	0.856	0.173	0.999	0.692	0.066	0.381
$p = 5$	min	14.0	14.0	2.0	0.392	0.003	0.994	0.002	0.0	0.234
	max	15364.0	6024.0	44.0	1.0	0.429	1.0	0.843	0.16	0.5
	mean	1211.467	535.267	9.533	0.797	0.104	0.998	0.64	0.061	0.426

the median coverage error $\mathrm{ME}(\mathcal{Y}_S)$ across all classes was approximately $\frac{1}{11}\sum_{i=1}^{11}\mathrm{ME}(\mathcal{Y}_{S_i}) \approx 0.01036$, which is about 12% of the corresponding mean for $\mathrm{ME}(\mathcal{Y}_{ES})$, which was 0.086. Similarly, the average value of $\mathrm{MEAN}(\mathcal{Y}_S)$ was 0.01218, representing approximately 12.8% of the value for the extreme supported solutions.

Furthermore, the ε-indicator also confirms the quality advantage of the supported nondominated points as a representative approximation. The average values were $I_\varepsilon(\mathcal{Y}_S)$ was 1.004 and $I_\varepsilon(\mathcal{Y}_{ES})$ was 1.042. This small ε-value indicates that only a marginal vector multiplied with each nondominated point such that the solutions in $\mathcal{Y}_S$ (weakly) dominate the entire nondominated point set.

The higher mean ratio of $|\mathcal{Y}_S|/|\mathcal{Y}_N|$ with 43.1% may be explained by the fact that the considered graphs are relatively sparse, and only a few source and sink nodes are involved. As shown in Table 5.1, a clear trend can be observed when examining the

Table 5.8: Numerical results for bi-objective instances with correlated cost objectives. The classes again contains 15 instances.

Class $p = 2$		$\lvert\mathcal{Y}_N\rvert$	$\lvert\mathcal{Y}_S\rvert$	$\lvert\mathcal{Y}_{ES}\rvert$	$\frac{\lvert\mathcal{Y}_S\rvert}{\lvert\mathcal{Y}_N\rvert}$	$\frac{\lvert\mathcal{Y}_{ES}\rvert}{\lvert\mathcal{Y}_N\rvert}$	HVR($\mathcal{Y}_S$)	HVR($\mathcal{Y}_{ES}$)	CE($\mathcal{Y}_S$)	CE($\mathcal{Y}_{ES}$)
$n = 20, m = 80$	$\ell = 0$	80.467	42.2	7.0	0.6	0.113	0.994	0.803	0.026	0.272
	$\ell = 1$	64.6	29.067	5.333	0.632	0.157	0.993	0.664	0.02	0.344
	$\ell = 2$	23.267	14.933	4.133	0.83	0.388	0.994	0.711	0.017	0.284
	$\ell = 3$	7.867	7.4	2.133	0.984	0.652	1.0	0.807	0.001	0.172
$n = 50, m = 200$	$\ell = 0$	107.4	46.067	8.667	0.479	0.168	0.992	0.882	0.029	0.215
	$\ell = 1$	66.2	35.4	6.667	0.654	0.191	0.993	0.766	0.03	0.299
	$\ell = 2$	32.467	21.933	5.0	0.781	0.263	0.995	0.78	0.033	0.288
	$\ell = 3$	5.933	5.733	2.0	0.968	0.625	0.993	0.698	0.026	0.238
$n = 100, m = 400$	$\ell = 0$	165.867	50.133	13.8	0.438	0.121	0.996	0.927	0.029	0.174
	$\ell = 1$	79.867	32.267	7.8	0.551	0.15	0.99	0.804	0.039	0.242
	$\ell = 2$	47.6	24.133	6.0	0.766	0.279	0.991	0.76	0.029	0.311
	$\ell = 3$	10.0	7.867	3.133	0.875	0.56	0.993	0.808	0.048	0.242
$n = 200, m = 800$	$\ell = 0$	244.533	61.267	14.533	0.306	0.069	0.994	0.913	0.03	0.195
	$\ell = 1$	67.667	30.267	8.2	0.539	0.151	0.993	0.887	0.045	0.254
	$\ell = 2$	36.2	20.8	5.867	0.674	0.202	0.987	0.736	0.048	0.289
	$\ell = 3$	9.6	8.2	2.867	0.907	0.517	0.992	0.686	0.04	0.295
$n = 400, m = 1600$	$\ell = 0$	248.133	60.6	19.533	0.287	0.089	0.994	0.938	0.029	0.169
	$\ell = 1$	103.133	38.533	12.0	0.45	0.175	0.991	0.922	0.036	0.178
	$\ell = 2$	57.133	24.533	7.267	0.59	0.234	0.992	0.845	0.039	0.256
	$\ell = 3$	22.467	13.067	4.333	0.734	0.322	0.989	0.797	0.035	0.274

arc-to-node ratio. In all instance classes where the number of arcs was four times the number of nodes, i.e., classes $i \in \{2, 4, 6, 8, 10\}$, not only was the number of nondominated points significantly higher compared to the remaining classes $i \in \{1, 3, 5, 7, 9, 11\}$, where the number of arcs was only twice as high as the number of nodes, but also the ratio of supported nondominated points to the entire nondominated set decreased.

The results obtained from the second test class presented in Table 5.5 and visualized in Figure 5.3, where the maximum arc capacity is increased, confirm the initial expectations. The sets $\mathcal{Y}_S$ and $\mathcal{Y}_{ES}$ are identical for the instance class with a maximum capacity of one. In this case, even the extreme supported set $\mathcal{Y}_{ES}$ already provides high-quality approximations, which aligns well with the findings reported in Sayın [2024].

However, as the maximum arc capacity increases, both the number of nondominated points and supported nondominated points grow substantially. The ratio $|\mathcal{Y}_S|/|\mathcal{Y}_N|$ remains relatively stable across all capacity levels, whereas the ratio $|\mathcal{Y}_{ES}|/|\mathcal{Y}_N|$ decreases significantly. This behaviour matches the expectations; increasing the maximum arc capacity does not introduce new cycle structures in the residual graph, but those combinations go from one solution to another over augmenting the full residual capacity of this cycle, creating many more *intermediate* solutions. These additional intermediate solutions increase both supported and total nondominated points without significantly altering their relative proportion, whereas $|\mathcal{Y}_{ES}|/|\mathcal{Y}_N|$ decreases as expected.

Moreover, while the hypervolume ratio for $\mathcal{Y}_S$ steadily increases with higher capacities, remaining close to 1 in all instance classes, the opposite trend is observed for $\mathcal{Y}_{ES}$. The hypervolume ratio decreases, and its coverage error increases. In the final instance classes with $c^{\max} = 100$, the hypervolume ratio of $\mathcal{Y}_S$ is about 18% higher than that of $\mathcal{Y}_{ES}$, and the coverage error of $\mathcal{Y}_S$ is only about 13% of the error measured for $\mathcal{Y}_{ES}$. These results strongly indicate that in networks with high arc capacities, supported nondominated points

Table 5.9: Numerical results of OCVR for bi-objective instances. Each class again contain 15 instances.

Class $p = 2$		OCER($\mathcal{Y}_S$)	OCER($\mathcal{Y}_{ES}$)
$n = 20, m = 80$	min	0.322	0.25
	max	1.0	0.644
	mean	0.735	0.453
$n = 50, m = 200$	min	0.091	0.261
	max	1.0	1.0
	mean	0.687	0.429
$n = 100, m = 400$	min	0.264	0.237
	max	1.0	0.6
	mean	0.578	0.408
$n = 200, m = 800$	min	0.097	0.195
	max	0.838	0.506
	mean	0.481	0.327
$n = 400, m = 1600$	min	0.267	0.154
	max	0.8	0.4
	mean	0.491	0.278
Overall	min	0.091	0.154
	max	1.0	1.0
	mean	0.594	0.379

Table 5.10: Netgen parameters for the different classes with more supply, sources and sinks, given as in Raith and Ehrgott, 2009.

Name	n	m	Sources	Sinks	Supply	Transhipment sources	Transshipment sinks
N01	20	60	9	7	90	4	3
N04	40	120	18	14	180	9	7
N07	60	180	27	21	270	14	10
N12	80	400	35	38	350	17	14

($\mathcal{Y}_S$) yield better representations than the extreme supported points ($\mathcal{Y}_{ES}$). It is worth emphasizing that even a maximum arc capacity of 100 is still relatively small in practical terms, suggesting that these observed trends could be even more pronounced in larger, real-world networks.

Based on the findings presented in Table 5.7, it can be observed that even in higher dimensions, where the number of nondominated points increases significantly, the performance metrics remain relatively stable. Both the ratios of supported to total nondominated points and the hypervolume ratios show only minimal variation. While there is a slight increase in the coverage error for the sets $\mathcal{Y}_S$ with increasing dimension, it remains relatively small, with a mean value of just 0.061 even in the case of $p = 5$. These observations suggest that the effectiveness of supported nondominated points as high-quality representations persists even in higher-dimensional settings.

As shown in Table 5.8, where different levels of cost correlation ℓ are considered, it can be observed that as the correlation level increases, the number of nondominated points decreases, while the ratio of supported nondominated points to the total set steadily increases. This indicates that higher correlation among objectives tends to simplify the

Table 5.11: Numerical results for bi-objective instances with more supply, sinks, and sources. The classes are similar to the one from Raith and Ehrgott [2009]. The classes again contain 15 instances.

Class $p = 2$		$\lvert\mathcal{Y}_N\rvert$	$\lvert\mathcal{Y}_S\rvert$	$\lvert\mathcal{Y}_{ES}\rvert$	$\frac{\lvert\mathcal{Y}_S\rvert}{\lvert\mathcal{Y}_N\rvert}$	$\frac{\lvert\mathcal{Y}_{ES}\rvert}{\lvert\mathcal{Y}_N\rvert}$	HVR($\mathcal{Y}_S$)	HVR($\mathcal{Y}_{ES}$)	CE($\mathcal{Y}_S$)	CE($\mathcal{Y}_{ES}$)
N01	min	29.0	19.0	7.0	0.068	0.02	0.948	0.805	0.023	0.098
	max	353.0	54.0	18.0	0.655	0.31	0.999	0.98	0.084	0.361
	mean	162.867	33.867	12.267	0.258	0.098	0.988	0.918	0.043	0.174
N04	min	120.0	30.0	15.0	0.103	0.033	0.99	0.933	0.021	0.068
	max	819.0	91.0	30.0	0.271	0.133	0.997	0.982	0.051	0.274
	mean	388.533	58.4	22.4	0.175	0.069	0.993	0.965	0.029	0.115
N07	min	452.0	66.0	33.0	0.08	0.03	0.993	0.973	0.018	0.048
	max	1455.0	117.0	49.0	0.195	0.086	0.996	0.987	0.03	0.124
	mean	736.067	85.2	40.067	0.123	0.058	0.995	0.982	0.023	0.073
N12	min	1192.0	96.0	65.0	0.063	0.034	0.995	0.984	0.01	0.028
	max	2577.0	188.0	95.0	0.099	0.059	0.998	0.995	0.023	0.06
	mean	1895.533	142.867	79.667	0.076	0.043	0.997	0.992	0.017	0.045

Table 5.12: Numerical results for same instances as in Table 5.12 but with 3 objectives. The class again contain 15 instances.

Class $p = 3$		$\lvert\mathcal{Y}_N\rvert$	$\lvert\mathcal{Y}_S\rvert$	$\lvert\mathcal{Y}_{ES}\rvert$	$\frac{\lvert\mathcal{Y}_S\rvert}{\lvert\mathcal{Y}_N\rvert}$	$\frac{\lvert\mathcal{Y}_{ES}\rvert}{\lvert\mathcal{Y}_N\rvert}$	HVR($\mathcal{Y}_S$)	HVR($\mathcal{Y}_{ES}$)	CE($\mathcal{Y}_S$)	CE($\mathcal{Y}_{ES}$)
N01	min	512.0	159.0	28.0	0.075	0.009	0.983	0.828	0.072	0.122
	max	8443.0	769.0	125.0	0.311	0.055	0.998	0.971	0.262	0.262
	mean	3647.467	438.933	60.4	0.154	0.022	0.991	0.915	0.119	0.209

structure of the Pareto front, making it easier to represent the solution set through supported nondominated points. Notably, for $\ell = 3$ the supported points constitute the dominant share of the nondominated set across all instance classes, and in many cases only very few nonsupported solutions remain. Hence, at high correlation levels, the Pareto front becomes almost entirely supported, implying that weighted-sum scalarization can capture most of the front with only minor loss of accuracy. These findings highlight the significant impact of objective correlation on the complexity of multi-objective solution spaces and confirm that highly correlated objectives lead to more other types of Pareto fronts. For each correlation level ℓ, the supported nondominated points reach an almost complete hypervolume approximation and small coverage errors, while maintaining comparatively low coverage errors, underlining their suitability as compact representatives of the full nondominated set.

Based on the results presented in Table 5.9, several observations can be made. Across all instance classes, the mean values of OCER($\mathcal{Y}_S$) range from approximately 0.491 to 0.74, indicating that the supported nondominated points achieve a large fraction of the optimal coverage attainable with the same number of points. This confirms that supported solutions provide a strong and reliable coverage-oriented approximation of the Pareto front. In contrast, the mean values of OCER($\mathcal{Y}_S$) are clearly lower. This suggests that instead of considering only extreme supported points, it may be beneficial to include other nondominated points to enhance representation quality.

For the test classes N01, N04, N07, and N12, Table 5.11 and Table 5.12 show that, as expected, the number of nondominated points increases when the total amount of supply, source nodes, and sink nodes is increased. The instances with three objectives illustrate more significantly the rapid growth in the number of nondominated points. For instance, the mean number of nondominated points for N01 was about 3647.5, while for N04, the mean increased drastically to 37,100.53 across the 15 instances, which is already more than twenty times larger than the mean of 162.9 observed for N01 in the bi-objective setting. This increase in nondominated points made the computation for the remaining classes, N07 and N12, computationally too demanding.

Additionally, as anticipated, the ratio of noticeably $|\mathcal{Y}_S|/|\mathcal{Y}_N|$ is smaller compared to the ratios observed in the previous test classes. The mean ratio across all four problem classes ranges only from 0.076 to 0.258, indicating that the supported set represents a comparatively small fraction of all nondominated solutions. However, despite this lower ratio, $\mathcal{Y}_S$ still provides high-quality representations. In the bi-objective cases shown in Table 5.11, the mean of HVR($\mathcal{Y}_S$) is 99.3%, and the mean coverage error remains minor at 0.028. This indicates that $\mathcal{Y}_S$ represents the solution set with high quality, maintaining a near-perfect hypervolume and low coverage error, even when it represents only a small subset of it.

Interestingly, while HVR($\mathcal{Y}_{ES}$) increases to 96.4% in these instances, it remains smaller than that of $\mathcal{Y}_S$. The mean coverage error for $\mathcal{Y}_{ES}$ 0.102, which is about 3.6 larger than the coverage error for $\mathcal{Y}_S$. These results suggest that even in instances with higher supply and more complex data structures, where the ratio of supported nondominated points is quite small, $\mathcal{Y}_S$ still represents the solutions at a very high quality. The high hypervolume ratio and minimal coverage error demonstrate that $\mathcal{Y}_S$ remains superior to $\mathcal{Y}_{ES}$ as a representation.

Thus, even when the problem instances become more complex, with an increasing number of nondominated points and smaller ratios, the representation provided by $\mathcal{Y}_S$ remains highly effective and reliable, with HVR($\mathcal{Y}_S$) close to 1 and a minimal coverage error.

In total, the results demonstrate that supported nondominated points $\mathcal{Y}_S$ consistently yield high-quality representations across all considered instances, especially in network settings with higher arc capacities. Unlike the extreme supported subset $\mathcal{Y}_{ES}$, which performs well in binary problems, $\mathcal{Y}_S$ significantly outperforms in terms of hypervolume and coverage error when the problem complexity increases. These findings highlight that the structural properties of the network, such as sparsity, arc capacity, and cost correlation, strongly influence the representation quality, with $\mathcal{Y}_S$ adapting much better across diverse scenarios. Thus, supported nondominated points provide a robust representation even in high-dimensional or densely structured MOIMCF problems.

5.4 Conclusion

The key finding of this chapter is the role of supported nondominated points as high-quality representations of the complete nondominated point set in MOIMCF. Across various test instances, supported solutions consistently demonstrated superior representational quality,

as measured by hypervolume ratio and coverage error. For all instances, the hypervolume ratio of the supported nondominated points as representations always are close to one and provides minor coverage errors. In contrast, extreme supported nondominated points yielded significantly lower quality measures, notably as arc capacities increased. Unlike the extreme supported subset $\mathcal{Y}_{ES}$, which performs well in binary problems, $\mathcal{Y}_S$ significantly outperforms in terms of hypervolume and coverage error when the problem complexity increases. These findings highlight that the structural properties of the network, such as sparsity, arc capacity, and cost correlation, strongly influence the representation quality, with $\mathcal{Y}_S$ adapting much better across diverse scenarios. Thus, supported nondominated points provide a robust representation even in high-dimensional or densely structured MOIMCF problems.

Beyond their representational advantages, the supported nondominated are generally more straightforward to compute than nonsupported ones and can serve as a foundation for two-phase methods. Although supported nondominated points provide high-quality representations, their number can still be numerous and sometimes constitute a significant part of the complete nondominated point set in MOIMCF problems. Future research could explore whether a strategically chosen subset of the supported nondominated points, potentially derived from intermediate solutions, could provide sufficiently high-quality representations while reducing computational effort. However, as indicated by the mean values of $\text{OCER}(\mathcal{Y}_S)$, adding unsupported nondominated points might sometimes be beneficial instead of supported ones. Hence, new general representation methods for MOIMCF problems could be of interest.

Additionally, it would be valuable to investigate whether strategically chosen nonsupported nondominated points might enhance the representation quality. As observed in binary problems like the binary knapsack and binary MOIMCF, extreme supported nondominated points alone already provide high-quality representations. It would be particularly interesting to evaluate the effectiveness of extreme supported or supported nondominated points in other MOCO problems, especially other network flow problems, and to derive criteria for identifying when supported nondominated points are necessary for high-quality representations or whether extreme supported nondominated points are sufficient.

6 Finding all Minimum Cost Flows

This Chapter addresses the problem of determining all optimal integer solutions of a linear integer network flow problem, referred to as the *all optimal integer flow (AOF)* problem. An algorithm with a time complexity of $\mathcal{O}(|F|(m+n)+mn+\zeta)$ is derived to determine the set F of all optimal integer flows in a directed network with n nodes and m arcs, where ζ represents the time required to find a single minimum cost flow.

It is worth noting that if the well-known method presented in Hamacher [1995] for finding the k best integer flows is stopped at the first sub-optimal flow, the resulting algorithm for solving the AOF problem has a running time of $\mathcal{O}(|F|m(n\log n+m)+\zeta)$. Our improvement is essentially made possible by replacing the shortest path sub-problem with a more efficient way to determine a so-called *proper zero cost cycle* using a modified depth-first search technique.

Additionally, the analysis provides an enhanced algorithm for determining the k best integer flows, which runs in $\mathcal{O}(kn^3+\zeta)$. Furthermore, lower and upper bounds for the number of all optimal integer and feasible integer solutions are established based on the fact that any optimal solution can be derived from an initial optimal *tree solution* combined with a conical combination of incidence vectors of all *induced cycles* with bounded coefficients.

The results in this Chapter were obtained in collaboration with Daniel Schmidt and Christane Spisla and have been published in Könen et al. [2022a]. Note that some ideas in this chapter are also based on Könen [2019].

6.1 Introduction

Being a fundamental and abstract model, minimum cost flows tend to ignore certain aspects of applications. For instance, drawing a planar graph on a rectangular grid with a minimum number of edge bends can be modeled as a minimum cost flow problem [Battista et al., 1998; Mendonça Neta et al., 2012; Jünger and Mutzel, 2004]. The result of this bend-minimization process affects indirectly and not predictably other properties of the final drawing, like the drawing area, the aspect ratio, or the total edge length. Consequently, different minimum cost flows can yield graph drawings all, with the minimum number of bends but completely different appearances.

The above criteria (and others) define the readability and *goodness* of a drawing, but eventually, whether a given drawing is good lies in the eye of the beholder. In situations like these, devising algorithms is a non-trivial task; particularly if no formal model for adherence exists and recognizing suitable solutions is up to the user. Here, a quick hands-on answer is to enumerate *all* minimum cost flows, i.e., all possibilities for a bend minimal drawing, and let the user decide on the details.

Christofides and Valls [1986] proposes an algorithm that enumerates all *basic* integer minimum cost flows. However, the algorithm may enumerate the same solution multiple times, prompting the need for a more efficient enumeration procedure. Here, the best we can hope for is an algorithm whose worst-case running time is polynomial in n, m *and* the number of enumerated flows, which generally, might be exponential in n and m. To ensure that each solution is only enumerated once, Hamacher [1995] proposes a binary partition approach and considers the k best integer flow problem: If $f^1, f^2, \dots$ are all feasible flows in some order of non-decreasing cost, Hamacher's algorithm will return $f^1, \dots, f^k$ in time $\mathcal{O}\big(km(n \log n + m) + \zeta\big)$, where ζ is the time needed to find *one* minimum cost flow. Sedeño-Noda and Espino-Martín [2013] propose another algorithm with the same running time but a reduced memory space requirement of $\mathcal{O}(k + m)$. Both algorithms essentially reduce to solving $\mathcal{O}(km)$ shortest path problems.

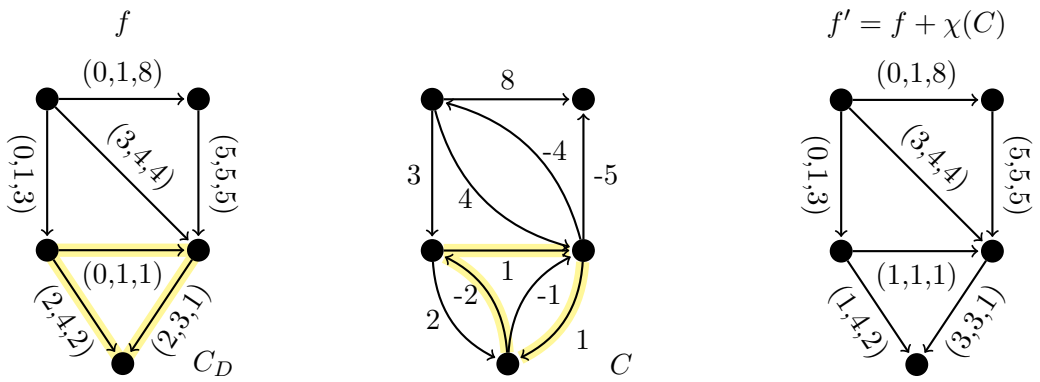
This chapter proposes an algorithm that finds *all* $|F|$ *many optimal integer* flows, i.e., all minimum cost flows, in output-polynomial time $O(|F|(m + n) + mn + \zeta)$, where F is the set of all optimal flows. Notably, stopping Hamacher's (or Sedeño-Noda and Espino-Martín's) algorithm at the first sub-optimal flow results in an algorithm with a running time of $\mathcal{O}(|F|m(n \log n + m) + \zeta)$. The key improvement here lies in replacing the shortest path sub-problem with a more efficient method for identifying a *proper zero cost cycle* using a modified depth-first search technique.

While alternative solutions play a pivotal role in multi-objective optimization, we will demonstrate how the resulting algorithm facilitates the identification of solution sets within this framework. It is a crucial part in the Algorithm to determine all supported efficient solutions which will be presented in Section 7.2.

In addition to the primary algorithm, this chapter presents an enhanced algorithm for the k best flow problem, running in $\mathcal{O}(kn^3 + \zeta)$, with the same space requirements as Hamacher's algorithm and compatible with Sedeño-Noda and Espino-Martín's improvements.

The relevance of finding all optimal (or the k best) solutions extends to various combinatorial optimization problems across different applications, as detailed in Hamacher [1995]. The determination of the k best flows has numerous applications, such as dynamic lot size problems [Hamacher, 1995], multi-objective minimum cost flow problems [Hamacher et al., 2007b], and solving minimum cost flow problems with additional constraints [Yan and Tu, 2002]. Further examples of the practical importance of identifying alternative optimal solutions can be found in Galle [1989] and Baste et al. [2019].

The organization of the chapter is as follows: Section 6.2 begins by introducing the problem and most important definitions for the subsequent chapters. Section 6.3 takes a given minimum cost flow and shows how to obtain another one with a modified DFS based procedure, while Section 6.4 presents the algorithm for the all optimal flow problem. Section 6.5 uses the algorithm to derive a procedure for the k best flow problem. Finally, upper and lower bounds on the number of feasible and optimal flows are discussed in Section 6.6.



$D = (V, A)$ (a) $D_f = (V, A_f)$ (b) $D = (V, A)$ (c)

Figure 6.1: Example of an augmentation on a 0-cycle with one unit. (a) Graph D, the arcs of which are labeled with (f_{ij}, u_{ij}, c_{ij}). Here, the lower bound $l_{ij} = 0$ for all arcs $(i, j) \in A$. The 0-cycle C_D is marked in yellow. (b) The residual graph D_f with $c_{ij}(f)$ as arc labels and the cycle $C \subset D_f$ in yellow, where C_D is the equivalent cycle in D. (c) The graph D with new flow values $f' = f + \chi(C)$.

6.2 Problem Definition and Notation

Definition 6.1. *The* all optimal integer flow problem *(AOF problem) consists of determining all integer flows which are optimal.*

After Property 2.66 it holds, that two feasible integer flows only differ on directed cycles in the residual graph, i.e., if f and f' are two feasible integer flows, then it holds:

$$f' = f + \sum_{j=1}^{k} \lambda_j \chi(C_j),$$

$$c(f') = c(f) + \sum_{j=1}^{k} \lambda_j c(f, C_j)$$

for directed cycles $C_1, \ldots, C_k$ in D_f and integers $\lambda_1, \ldots \lambda_k > 0$. In particular, two optimal integer flows f and f' can only differ on directed cycles C_j in D_f with zero cost.

Definition 6.2. *A* 0-cycle *is a zero cost cycle $C \subset D_f$, i.e., a cycle $C \subset D_f$ with $c(f, C) = 0$. We call a 0-cycle $C \subset D_f$* proper *if for all arcs $(i, j) \in C$ we have $(j, i) \notin C$.*

Then, obtaining a second optimal flow seems to be very simple: Identify a 0-cycle $C \subset D_f$ and augment f along C by λ as defined above, i.e., $f' = f + \lambda\chi(C)$ (see, Figure 6.1 for an example).

Unfortunately, augmenting f along a 0-cycle C does not necessarily yield a *different* flow f'; for instance, we have $f = f'$ if we augment along $C = ((i,j),(j,i))$. In order to avoid this technical problem, we only consider proper 0-cycles $C \subset D_f$.

In the exposition of this thesis, we assume that D contains no anti-parallel arcs, i.e., if $(i,j) \in A$ then $(j,i) \notin A$, to avoid any confusion with the notation. However, the only difficulty with multiple arcs between a pair of nodes is to keep track of the relation that links residual arcs with original arcs, which can be done as mentioned in Sedeño-Noda and Espino-Martín [2013]. It is straightforward to modify the algorithms in further sections to deal with graphs that contain multiple arcs between a pair of nodes. Note that a proper cycle is then defined as a cycle C that does not contain a pair of *symmetric* arcs, i.e., $\{(i,j),(j,i)\} \in D_f$ where both refer to the same arc $(i,j) \in D$. After Property 2.69 it holds, that the cost of a cylce C in D_f

$$c(f,C) = \overline{c}(f,C).$$

Therefore, it makes no difference whether we search for cycles in D_f with zero cost or zero reduced cost. Again we assume that any node potential π is optimal, i.e., satisfying the complementary slackness optimality conditions and thus satisfying $\overline{c}_{ij}(f) \geq 0$ for all $(i,j) \in D_f$.

6.3 Getting From one Minimum Cost Flow to Another

The idea behind the algorithm presented in Section 6.4 is to find proper 0-cycles efficiently. For this purpose, we will reduce the network D. To do so we will use a well-known property about the arcs in a proper 0-cycle. This result will help us reduce the network and obtain a flow problem in which every feasible solution will yield an optimal solution for the original MCF problem.

The following property is well-known (see, e.g., Christofides and Valls [1986] and Calvete and Mateo [1996]).

Property 6.3. *Let f and f' be two optimal integer flows. Then, f and f' only differ on arcs with $\overline{c}_{ij} = 0$. Or, in other words, if $f_{ij} \neq f'_{ij}$ then $\overline{c}_{ij} = 0$.*

Let $X := \{(i,j) \in A : \overline{c}_{ij} \neq 0\}$ be the set of arcs with non-zero reduced cost. We may remove all arcs in X and obtain a reduced network. However, it may be that the arcs in X carry flow. Consequently, we have to adjust the demand vector accordingly to maintain a feasible flow. Consider the reduced network $D' = (V, A')$ with $A' = A \backslash X$ and new flow balance values $b'_i = b_i - \sum_{(i,j)\in X} f_{ij} + \sum_{(j,i)\in X} f_{ji}$. We can easily limit any feasible flow in D to a feasible flow in D' by simply adopting all flow values. For a flow $f \in D$, we always refer by $f \in D'$ to the reduced flow where all flow values are copied for all arcs $a \in A'$. We denote by Γ' the set of all feasible integer flows of the reduced network D'. The fact that there is only a finite number of flows in Γ' follows from the assumption that there is no arc (i,j) with $u_{ij} = \infty$. For a feasible solution $f' \in \Gamma'$, we define

$$f^*(a) = \begin{cases} f'(a), & \text{if } a \in A', \\ f(a), & \text{if } a \in X. \end{cases}$$

The next Lemma demonstrates how every optimal flow in D can be obtained from the combination of an initial optimal flow and some feasible flow in D'.

Lemma 6.4. *Let f be an optimal integer solution. If f' is a feasible integer solution in D' and f^* is defined as above, then f^* is an optimal solution in D and vice versa.*

Proof.

$$
\begin{aligned}
& f' \text{ is feasible in } D' \\
\iff & f \text{ and } f' \text{ differ on a set of cycles in } D'_f && \text{(Property 2.66)} \\
\iff & f \text{ and } f' \text{ differ on a set of cycles in } D'_f \\
& \text{ with reduced cost equal to zero} && \text{(Definition of } D') \\
\iff & f \text{ and } f^* \text{ differ on a set of cycles in } D_f \\
& \text{with reduced cost equal to zero} && \text{(cycles in } D'_f \text{ also exist in } D_f) \\
\iff & f^* \text{ is optimal} && \text{(Optimal flows only differ} \\
& && \text{on zero cycles and Property 2.69)}
\end{aligned}
$$

□

Given a feasible flow f in D', we can compute another feasible flow by finding a proper cycle in the residual network of D'. Therefore, we can determine an additional optimal flow in D by finding a proper cycle in D'_f instead of finding a proper 0-cycle in D_f. This gives us the following theorem:

Theorem 6.5. *Let f be an optimal integer flow. Then every proper cycle in D'_f induces another optimal integer flow in D.*

Proof. Any cycle in D'_f has zero cost in D_f. □

We now show that we can determine such a proper cycle with the following depth-first search (DFS) technique. Recall the different arc types in a directed DFS-forest. *Tree arcs* are the arcs in the trees of the DFS-forest. *Backward arcs* are the arcs (i, j) connecting a vertex i to an ancestor j in the same DFS-tree. Non-tree arcs (i, j) are called *forward arcs* if the arc connects a vertex i to a descendant j in the same DFS-tree. All other arcs are called *cross arcs*.

In order to find a proper cycle in D', let Γ be a DFS-forest of D'. Recall that π_i is the predecessor of node $i \in T \subseteq \Gamma$ and $\text{dfs}(i)$ denote the DFS number (compare *discovery time* in Cormen et al. [2001]) of node i, i.e., the number when node i is visited during the DFS process. There cannot exist a cross arc (i, j) with $\text{dfs}(i) < \text{dfs}(j)$, i.e., a cross arc from a node i that is visited before the arc's sink node j. In other words, if we assume that the children of each node are drawn from left to right and the different trees of the DFS-forest are also drawn from left to right, a cross arc always goes from the right to the left. Therefore, there can be cross arcs connecting two different trees of the DFS-forest, but those arcs can never be included in a cycle. Then the following nice property hold.:

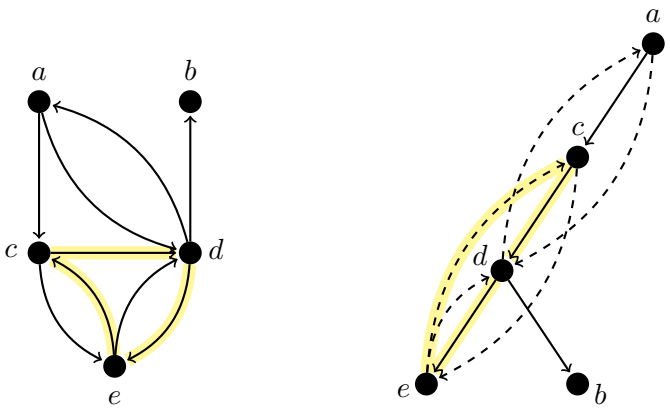


Figure 6.2: (a) The residual graph with a proper cycle marked in yellow and (b) the corresponding DFS tree with the proper backward cycle.

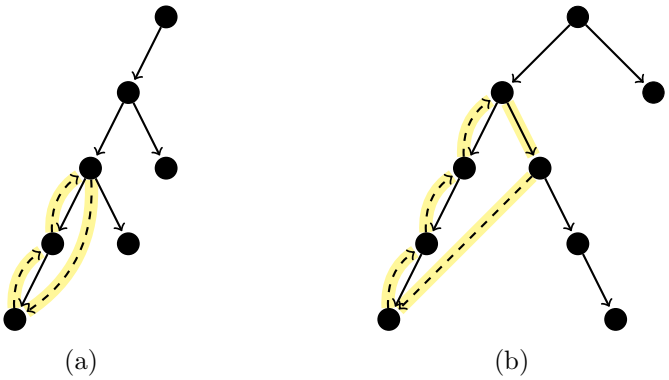


Figure 6.3: Proper cycles in a tree of Γ without containing a long backward arc. (a) A proper forward cycle and (b) a proper cross cycle.

Property 6.6 (Cormen et al., 2001)**.** *Each cycle C in D contains at least one backward arc and no cycle can contain an arc connecting two different trees of Γ, i.e., there is no arc $(i,j) \in C$ with $i \in T_l$ and $j \in T_k$ for $T_l, T_k \in \Gamma$ with $k \neq l$.*

We call a backward arc (i,j) with $\pi_i \neq j$ a *long backward arc*. Otherwise, we call it a *short backward arc*. If there exists a long backward arc, we immediately have determined a proper cycle formed by the long backward arc (i,j) and the distinct path from j to i in T (see Figure 6.2). However, even without the existence of such a backward arc, there can exist proper cycles, possibly containing a forward or a cross arc. See Figure 6.3 for examples. However, every proper cycle contains at least one cross, forward or long backward arc.

Definition 6.7. *Let Γ be a DFS-forest of a directed graph D. A* proper backward cycle *in D is a proper cycle which is formed by a long backward arc (i,j) and the unique path from j to i in the corresponding tree T.*

A proper forward cycle *in D is a proper cycle formed by a forward arc (i, j) and the path from j to i only consisting of short backward arcs.*
A proper cross cycle *in D is a proper cycle formed by a cross arc (i, j), the path from a to i in the tree T, where a is the first common ancestor of the nodes i and j in T, and the path from j to a only consisting of short backward arcs.*

Figure 6.2 and Figure 6.3 show examples of the above definitions. In the next step, we will show that it is enough to search only for these kinds of proper cycles to detect one proper cycle.

Lemma 6.8. *If there exists a proper cycle in D'_f, then there also exists a proper backward, forward or cross cycle.*

Proof. Let C be a proper cycle in D'_f. We know that every cycle contains at least one backward arc and cannot contain arcs connecting different trees of the forest.

Case 1: There is at least one long backward arc in C. In this case, we have a proper backward cycle.

Case 2: There is no long backward arc in C, but at least one cross arc. To simplify, we can replace all forward arcs by tree arcs so that our cycle has only cross arcs, tree arcs and short backward arcs. Now, let l be the node with the least depth in C and let C start with node l. Further, let (i, j) be the last used cross arc in C. We want to show that a path made from short backward arcs from j to the lowest common ancestor a of i and j exists in the DFS-forest. Then, together with the tree arcs-path from a to i, we have a proper cross cycle. Since (i, j) is the last cross arc in C before returning to l, we have a path from j to l only consisting of short backward arcs. Suppose that a has less depth than l. It follows that i is in a different sub-tree than the one rooted at l. To get from l to i, cycle C must contain a cross arc leading to the sub-tree of i, because in C we cannot change sub-trees by going up the tree above the depth of l. The definition of cross arcs gives us $\text{dfs}(i) < \text{dfs}(l)$ and $\text{dfs}(j) < \text{dfs}(i)$ and we have $\text{dfs}(l) < \text{dfs}(j)$ because of the backward arcs-path from j to l – a contradiction. So a must have a greater or equal depth than l and therefore is reachable from j by short backward arcs.

Case 3: The cycle C contains no long backward arcs, no cross arcs but at least one forward arc. The only other kinds of arcs left are tree arcs and short backward arcs. Cycle C must contain short backward arcs, especially the ones from the sink of a forward arc to its source to close the cycle. Thus, we have at least one proper forward cycle. In fact, C cannot contain tree arcs since it is proper and we have no long backward arcs.

□

6.4 The All Optimal Integer Flow Algorithm

This section presents an algorithm that, given an initial optimal integer solution f to the MCF, determines the set F of all optimal integer flows in $\mathcal{O}(|F|(m+n)+mn)$ time. First,

we specify how to find an optimal flow different from the given one. Then, we show how we can partition the solution space of the AOF problem in D, given two optimal solutions. This gives us two smaller AOF problems and, recursively, we can find all optimal integer flows.

First, we provide an algorithm to determine a second feasible flow by determining one proper cycle of the aforementioned kinds. Herefore, we define a variable *SBAlow(i)* for each node $i \in V$ in the DFS-forest defined as the dfs number of the node closest to the root reachable from i only using short backward arcs. We may compute the *SBAlow* value analogously to the lowpoint as in Hopcroft and Tarjan [1973] during the DFS-procedure: Whenever the DFS visits a node i, we check whether there is a short backward arc (i, π_i). If so, we set $\mathit{SBAlow}(i) = \mathit{SBAlow}(\pi_i)$; otherwise, $\mathit{SBAlow}(i) = \text{dfs}(i)$. With these variables, we can quickly check if a path exists that only consists of short backward arcs from one node to another. This will help us to decide if proper forward or cross cycles exist.

We use Lemma 6.8, which guarantees that if any proper cycle exists, there is either a proper forward, backward or cross cycle. Therefore, it suffices to check for these three kinds of cycles only. If one long backward arc exists, we have already determined a proper backward cycle. For any forward arc (i, j), we test if a path from j to i exists only containing short backward arcs by checking if $\mathit{SBAlow}(j) \leq \text{dfs}(i)$. In this case, we know that there exists a path from j to i only consisting of short backward arcs, and hence we have determined a proper forward cycle. For any cross arc (i, j) we have to test if $\mathit{SBAlow}(j) \leq \text{dfs}(a)$, where a is the first common ancestor of i and j. If so, we know that a path from j to a exists only containing short backward arcs. In this case, we have determined a proper cross cycle. The following Algorithm 7 determines a proper cycle if one exists and in this case returns a second feasible solution.

Lemma 6.9. *Given a feasible integer flow f in D, Algorithm 7 determines another feasible integer flow different from f in time $\mathcal{O}(m+n)$ or decides that no such flow exists.*

Proof. The algorithm finds one proper backward, forward or cross cycle, if one exists, and therewith constructs a feasible flow distinct from f. The correctness follows from Theorem 6.5 and Lemma 6.8.

We can create the residual network and a DFS-forest in time $\mathcal{O}(m+n)$. Determining the kinds of arcs (tree, forward, backward or cross arcs) can also be accomplished in $\mathcal{O}(m+n)$ time: During the computation of the DFS-forest, we mark all tree arcs. If for a non-tree arc (i, j) we have $\text{dfs}(i) < \text{dfs}(j)$, then (i, j) is a forward arc. Otherwise, we check the lowest common ancestor of i and j. If it is j, we have a backward arc, otherwise (i, j) is a cross arc. The lowest common ancestor queries can be answered in constant time provided a linear preprocessing time [Harel and Tarjan, 1984]. The determination of the *SBAlow* values can also be accomplished during the DFS-procedure in $\mathcal{O}(m+n)$ time.

Let $m = m_1 \dot\cup m_2 \dot\cup m_3 \dot\cup m_4$ be a partition of m, where m_1 are all tree arcs, m_2 all backward arcs, m_3 all forward arcs and m_4 all cross arcs in the DFS forest. We test for all backward arcs (i, j) in $\mathcal{O}(1)$ time whether $\pi_i \neq j$. In this case, we encounter a long backward arc and we need additional $\mathcal{O}(n)$ time to create the proper backward cycle C_{ij}

Algorithm 7: FindAnotherFeasibleFlow

Data: feasible flow f in D
Result: Another feasible flow f' with $f' \neq f$, if one exists.

```
1  D_f ← BuildResidualNetwork(D)
2  Γ ← BuildDFSForest(D_f)              // Also determines all SBAlow values
3  Preprocess(Γ)                        // Determine forward, backward and cross arcs
4  for all backward arcs (i, j) do
5      if π_i ≠ j then
           // (i, j) is long backward arc
           // Build cycle from backward arc and j-i tree path
6          C_ij ← proper backward cycle (i, j) + j-i tree arcs path
7          f' ← f + χ(C_ij)
8          return f'
9  for all forward arcs (i, j) do
10     if SBAlow(j) ≤ dfs(i) then
           // Build cycle from forward arc and j-i short backward arcs path
11         C_ij ← proper forward cycle (i, j) + P_ji
12         f' ← f + χ(C_ij)
13         return f'
14 for all cross arcs (i, j) do
15     a ← LowestCommonAncestor(i, j)
16     if SBAlow(j) ≤ dfs(a) then
           // Build cycle from cross arc and j-i short backward arcs path
17         C_ij ← proper cross cycle (i, j) + P_ja + a-i tree arcs path
18         f' ← f + χ(C_ij)
19         return f'
   // No cycle found and therefore no other flow
20 return null
```

Algorithm 8: FindAnotherOptimalFlow

Data: optimal flow f in D and the reduced costs $\overline{c}$
Result: Another optimal flow f^* with $f^* \neq f$, if one exists.

```
// Reduce network
D' ← D
A' ← {(i,j) ∈ A | c̄_ij = 0}
b' ← AdaptDemandVector(f, D')
f' ← FindAnotherFeasibleFlow(f, D')
if f' = null then Stop
for all arcs a ∈ D do
    if a ∈ A' then f*(a) ← f'(a)
    else f*(a) ← f(a)
return f*
```

and adapt the flow. So, we need $\mathcal{O}(m_2)$ time for all backward arcs. For a forward arc, we also test in time $\mathcal{O}(1)$ if $SBAlow(j) \leq \text{dfs}(i)$. Overall, we need $\mathcal{O}(m_3)$ time for all forward arcs. As discussed before, the construction of a possible proper forward cycle and the returned flow take $\mathcal{O}(n)$ time. We also check if $SBAlow \leq \text{dfs}(a)$ in $\mathcal{O}(1)$ time for cross edges. The lowest common ancestor a is determined in $\mathcal{O}(1)$ time. Again, a proper cross cycle and the flow f' can be created in $\mathcal{O}(n)$ time. Altogether, we have a run time of $\mathcal{O}(m_2) + \mathcal{O}(m_3) + \mathcal{O}(m_4) + \mathcal{O}(n)$. The last term corresponds to the construction of C_{ij} and f'. Thus, the algorithm determines another feasible integer flow in $\mathcal{O}(n+m)$ or decides that none exists. □

However, there is no need to first build the DFS-forest and then test if a proper cycle exists for the different arc sets. We may also integrate all checks during the DFS-procedure, ending with a probably faster algorithm in practice. We choose the presented formulation of Algorithm 7 for better readability.

Next we describe how we use Algorithm 7 to determine another optimal flow in a network. Due to Lemma 6.4, any feasible flow in the reduced network D' is an optimal flow in D. Therefore, we only need to reduce the network and compute a second feasible flow to determine another optimal flow different from an initial optimal flow. Algorithm 8 follows this procedure.

Lemma 6.10. *Given an initial optimal integer flow f in D and the reduced costs $\overline{c}$, Algorithm 8 determines another optimal integer flow different from f in $\mathcal{O}(m+n)$ time or decides that no such solution exists.*

Proof. The algorithm reduces the network D following the description from the paragraph prior to Lemma 6.4. Thereafter, it computes another feasible flow f' in the reduced network given an optimal flow f in D, which is naturally feasible in D'. Then, f' can be transformed into another optimal flow f^* in D. The correctness of Algorithm 8 follows from Lemma 6.4.

Algorithm 9: FindAllOptimalFlows

Data: optimal flow f in D and the reduced costs $\overline{c}$
Result: All optimal integer flows in D

```
1  f* ← FindAnotherOptimalFlow(f, c̄, D)
2  if f* = null then Stop
3  Print(f*)
   // Partition solution space and find new optimal flow
4  a ← arc a with f(a) ≠ f*(a)
5  if f(a) < f*(a) then
6  |  FindAllOptimalFlows(f, c̄, D) with u_a = f(a)
7  |  FindAllOptimalFlows(f*, c̄, D) with l_a = f(a) + 1
8  else
9  |  FindAllOptimalFlows(f, c̄, D) with l_a = f(a)
10 |  FindAllOptimalFlows(f*, c̄, D) with u_a = f(a) − 1
```

We can reduce the network in $\mathcal{O}(m)$ time, and due to Lemma 6.9, determine another feasible flow in the reduced network in $\mathcal{O}(m' + n)$ time using the initial flow f as input. Here, m' denotes the number of arcs in D'. Therefore, we can find another optimal solution in time $\mathcal{O}(m + n)$ or decide that none exists. □

Since we can compute a second optimal flow in D, we can now apply the binary partition approach as used in Hamacher [1995] or Sedeño-Noda and Espino-Martín [2013]. Let F be the set of optimal flows in D, which we also call the *optimal solution space*, and let $f_1, f_2 \in F$. Algorithm 9 determines all optimal flows, by recursively dividing F into F_1 and F_2 such that f_1 and f_2 are optimal flows in F_1 and F_2, respectively, and then seeks more flows in F_1 and F_2. This can easily be done by identifying some arc (i, j) with $f_{1,ij} \neq f_{2,ij}$. Such an arc must exist since $f_1 \neq f_2$. Assume w.l.o.g. that $f_{1,ij} < f_{2,ij}$. We set

$$F_1 := \{z \in F : z_{ij} \leq f_{1,ij}\}$$

and

$$F_2 := \{z \in F : z_{ij} \geq f_{1,ij} + 1\}.$$

This partitioning implies that each of the new minimum cost flow problems is defined in a modified flow network with an altered upper or lower capacity of a single arc (i, j). By dividing the new solution spaces again until no more other optimal flow exists, all optimal solutions are found, whose number we denote by $|F|$. Our main result follows:

Theorem 6.11. *Given an initial optimal integer solution f and the reduced costs $\overline{c}$, Algorithm 9 solves the AOF problem in $\mathcal{O}(|F|(m + n))$ time, where F is the set of all optimal integer flows.*

Proof. In each recursive call of Algorithm 9, a new optimal integer flow different from the input one is found if one exists. Now let F_i be the current optimal solution space in an algorithm call and let F_j and F_k be the optimal solution spaces of the two following

calls. Every optimal flow is found since $F_i = F_j \cup F_k$ and no flow is found twice because $F_j \cap F_k = \emptyset$.

We will now show that we need $\mathcal{O}(|F|)$ calls of Algorithm 9 to find all optimal integer flows. We associate each algorithm call with one optimal flow. If we find another optimal solution, we associate this call with the newly found flow. Otherwise, we associate the call with the input flow. Because each flow induces exactly two more calls, there are at most three calls associated with one optimal flow. The primary time effort of Algorithm 9 is the computation of another optimal flow (Algorithm 8), which takes $\mathcal{O}(m+n)$ time. We do not have to determine the reduced cost again, since changing the flow value on arcs with zero reduced cost maintains their reduced costs. So in total, we solve the AOF problem in $\mathcal{O}(|F|(m+n))$ time. □

6.4.1 Remarks

The initial optimal integer flow may be obtained by using, for example, the network simplex algorithm or the enhanced capacity scaling algorithm [Ahuja et al., 1993; Orlin, 1993]. The latter determines an optimal flow in $\mathcal{O}((m \log n)(m + n \log n))$, i.e., in strongly polynomial time Orlin [1993].

Since we can compute the node potential as described in Section 2.5.2 by a shortest path problem in D_f, possibly containing arcs with negative cost, we need $\mathcal{O}(mn)$ time to determine the nodes' potentials. Given these potentials, we can determine the reduced costs in $\mathcal{O}(m)$ time. Then, an algorithm for the determination of all $|F|$ many integer flows needs $\mathcal{O}(|F|(m+n) + mn)$ time.

Using the network-simplex algorithm yields a so-called basic feasible solution in which an optimal node potential can be obtained in $\mathcal{O}(n)$ time (see Ahuja et al. [1993]). Then, an algorithm for the determination of all $|F|$ many integer flows reduces to $\mathcal{O}(|F|(m+n))$ time. Sedeño-Noda and Espino-Martín introduce techniques in Sedeño-Noda and Espino-Martín [2013] to reduce the memory space of the partitioning algorithm. With these techniques, it is possible to implement Algorithm 9 to run in space $\mathcal{O}(|F|+m)$. Furthermore, it is not necessary to reduce the graph at each call of Algorithm 8. Instead, it is sufficient to reduce the network D once in the beginning and to determine all feasible solutions in the same network D' because arcs with zero reduced cost maintain their reduced costs. However, this does not affect the asymptotical running time. We shall use this implementation of Algorithm 8 in the next chapter. The algorithm may also be used to determine all circulations, all feasible flows and all s-t flows with given value ϕ since these flow problems may be transformed into each other.

6.5 Improved Running Time for the k-Best Flow Problem

In the present section, we will improve the k best flow algorithm by Hamacher [1995] and derive a new one with running time $\mathcal{O}(kn^3)$ or $\mathcal{O}(k(n^2 \log n + mn))$ for sparse graphs, instead of $\mathcal{O}(k(mn \log n + m^2))$.

Definition 6.12 (Sedeño-Noda and Espino-Martín, 2013)**.** The k best integer flow (KBF) *problem consists of determining the k best integer solutions of the MCF problem. In other words, identifying k different integer flows f_i with $i \in \{1, \dots, k\}$ such that $c(f_1) \leq c(f_2) \leq \dots \leq c(f_k)$ and there is no other integer flow $f_p \neq f_i$ with $i \in \{1, \dots, k\}$ with $c(f_p) < c(f_i)$.*

As in Hamacher's algorithm [Hamacher, 1995], we want to determine a second optimal flow by determining a proper cycle of minimal cost. Using results from the previous sections, we can find this cycle more efficiently. Recall that we obtain a second-best flow by augmenting an optimal flow over a proper minimal cycle with one flow unit. After this, we will use the binary partition approach, as presented in Hamacher [1995] or Sedeño-Noda and Espino-Martín [2013].

Hamacher introduces an idea to improve the average complexity of his algorithm: Let $\overline{D} = (\overline{d}_{ji})$ be the distance table of D_f concerning $\overline{c}(f)$, i.e., $\overline{d}_{ji}$ is the length of the shortest path P_{ji} from j to i in D_f with length $\overline{c}(f, P_{ji})$. The distance table may be computed in time $\mathcal{O}(n^3)$ using the Floyd-Warshall algorithm or in time $\mathcal{O}(n^2 \log n + mn)$ by (essentially) repeated calls to Dijkstra's algorithm; the latter is more efficient on sparse graphs.

Definition 6.13 (Hamacher, 1995)**.** *We define a* proper (i, j)-cycle *$C \in D_f$ as a proper cycle that contains the arc (i, j). In addition, a proper (i, j)-cycle $C \in D_f$ is called* minimal *if it has minimal cost overall proper (i, j)-cycles.*

Hamacher shows the following property for proper minimal (i, j)-cycles:

Property 6.14 (Hamacher, 1995)**.** *For any arc $(i, j) \in A_f$ with $(j, i) \notin A_f$, i.e., for any arc with no anti-parallel arc in A_f, the cost of a proper minimal (i, j)-cycle $C \in D_f$ is given by $c(f, C) = \overline{c}_{ij}(f) + \overline{d}_{ji}$.*

When using Hamacher's idea, the problem is that it only applies to arcs with no anti-parallel arc in A_f. For all other arcs, the cost of the corresponding proper minimal (i, j)-cycle has to be computed by finding a shortest path in $D_f \setminus \{(j, i)\}$. In the following result, we show that we can use the previously obtained Algorithm 9 and the complementary slackness optimality conditions (Theorem 2.71) of an optimal solution to overcome this problem and restrict ourselves to consider only arcs with no anti-parallel arcs in D_f.

Lemma 6.15. *Given an initial optimal integer flow f, Algorithm 10 determines a second-best flow in $\mathcal{O}(n^3)$ time or decides that no such flow exists.*

Proof. Let f be an optimal integer solution. Assume that there exists another optimal solution f' in D. Then D_f contains at least one proper 0-cycle. Since we can compute one proper 0-cycle with Algorithm 8 by Lemma 6.10, we can determine a second-best integer flow.

So assume that there exists no other optimal flow different from f. Let C be a minimal proper cycle. No proper 0-cycle can exist in D_f due to Theorem 6.5. So $\overline{c}(C, f) > 0$, because there cannot be a cycle with negative costs due to the negative cycle optimality conditions. Therefore, there exists an arc $(i, j) \in C$ with $\overline{c}_{ij}(f) > 0$. Since f is an optimal solution, the complementary slackness optimality conditions (Theorem 2.71) give us that $\overline{c}_{ij}(f) > 0$ only holds if $(i, j) \in D$ and $f_{ij} = l_{ij}$ or $(j, i) \in D$ and $f_{ji} = u_{ji}$. Therefore, we

Algorithm 10: FindSecondBestFlow

Data: optimal flow f in D
Result: A second-best flow f' with $f' \neq f$, if one exists.

```
1  y ← ComputeNodePotential(f, D)
2  c̄ ← ComputeReducedCost(y, D)
3  f' ← FindAnotherOptimalFlow(f, c̄, D)
4  if f' = null then
5      (D̄, P) ← DetermineDistanceTableAndPaths(c̄, f, D)
6      Ā ← {(i,j) ∈ D_f : (i,j) ∈ D with f_ij = l_ij or (j,i) ∈ D with f_ji = u_ji}
7      C ← arg min{c(f,C) = c̄_ij + d̄_ji : C = {(i,j)} ∪ P_ji with (i,j) ∈ Ā}
8      if C = null then Stop
9      f' ← f + χ(C)
10 return f'
```

have that $(i,j) \in \overline{A} := \{(i,j) \in D_f : (i,j) \in D$ with $f_{ij} = l_{ij}$ or $(j,i) \in D$ with $f_{ji} = u_{ji}\}$ and $(j,i) \notin D_f$, (since the flow value of the corresponding arc of (i,j) in D is equal to the upper or lower capacity of this arc). Since $(j,i) \notin D_f$ it holds that $\overline{c}(C,f) = \overline{c}_{ij} + \overline{d}_{ji}$ due to Property 6.14. So Algorithm 10 finds a minimal proper cycle and hence a second-best optimal flow.

Now for the run time. Computing the node potentials by solving a shortest path problem in D_f as described in Section 2.5.2 takes $\mathcal{O}(mn)$ time because of possible negative arc cost. Given these potentials, we can determine the reduced costs in $\mathcal{O}(m)$ time. If there exists a proper 0-cycle, we can determine one in time $\mathcal{O}(m+n)$ by Lemma 6.10 or decide that no proper 0-cycle exists. After that, we compute the distance table in $\mathcal{O}(n^3)$ time. Notice that this approach also provides the paths P_{ji} in addition to the cost $\overline{d}_{ji}$. The minimal proper cycle is given by $\arg\min\{c(f,C) : C = \{(i,j)\} \cup P_{ji} : (i,j) \in \overline{A}\}$. Given the distances $\overline{d}_{ji}$ and the reduced costs $\overline{c}_{ij}(f)$ for all (i,j) in D_f we can determine $c(f,C) = \overline{c}_{ij} + \overline{d}_{ji}$ in $\mathcal{O}(1)$ and can determine the arg min in $\mathcal{O}(m)$ time.

As a result, we can compute a minimal proper cycle and, therefore a second-best integer flow in time $\mathcal{O}(n^3)$. □

Since we are able to find a second-best flow in $\mathcal{O}(n^3)$ time and by using the binary partition approach as in Hamacher [1995] or Sedeño-Noda and Espino-Martín [2013], we obtain the following result:

Theorem 6.16. *Given an optimal integer flow f, Algorithm 11 determines the k best integer solutions for a given MCF problem in time $\mathcal{O}(kn^3)$.*

Proof. The while loop iterates at most $k-1$ times. Due to Lemma 6.15, the second-best flow determination takes $\mathcal{O}(n^3)$ time in each iteration. The binary heap insertion and the extraction operations require $\mathcal{O}(m \log m)$ time [Sedeño-Noda and Espino-Martín, 2013]. So the algorithm needs $\mathcal{O}(kn^3)$ time overall. The uniqueness of any flow follows from the used partitioning approach [Hamacher et al., 2007b]. □

Algorithm 11: FindKBestFlows

Data: optimal flow f in D
Result: k best flows

```
1  i ← 1
2  f' ← FindSecondBestFlow(f, D)
   // Insert f and f' in binary heap with c(f') as key
3  if f' ≠ null then B ← B ∪ {(f, f', D)}
4  while B ≠ ∅ and i ≤ k do
5      i++
6      (f_i^1, f_i^2, D_i) ← ExtractMin(B)
7      Print(f_i^2)
8      if i = k then break
       // Partition solution space by adjusting upper and lower bounds so
          that f_i^1 is optimal flow in D_i^1 and f_i^2 is optimal flow in D_i^2
          (compare Algorithm 9)
9      {D_i^1, D_i^2} ← Partition(f_i^1, f_i^2, D_i)
10     f' ← findSecondBestFlow(f_i^1, D_i^1)
11     if f' ≠ null then B ← B ∪ {f_i^1, f', D_i^1}
12     f'' ← findSecondBestFlow(f_i^2, D_i^2)
13     if f'' ≠ null then
14         B ← B ∪ {f_i^2, f'', D_i^2}
```

6.6 Bounds on the Number of Feasible and Optimal Flows

In this section, we will introduce further theoretical results. We then use these results to give an upper and a lower bound for the number of all optimal and of all feasible integer flows. The main characteristics of the obtained bounds bases on the fact that any optimal solution can be obtained by an initial optimal *tree-solution* plus a conical combination of all incidence vectors of the *zero cost induced cycles* with bounded coefficients. Again, we only consider integer flows for the remaining of this thesis.

The general idea for the upper and lower bound is based on the fact that we can represent any undirected cycle C_D in D as a composition of some unique cycles C_{ij} induced by arcs $(i,j) \in C_D \setminus T$ (see Definition 2.74). This result is well-known and the base for several algorithms for finding a so-called fundamental set of cycles of a graph, see, e.g., Welch [1966]. Here, we define a composition of two sets of arcs as the symmetric difference of the two sets

$$A \circ B = (A \cup B) \backslash (A \cap B).$$

Recall that any simple path P (cycle C) in D_f corresponds to an undirected path P_D

(cycle C_D) in D and we had defined the incidence vector of P as

$$\chi_{ij}(P) := \begin{cases} 1, & \text{if } P \text{ traverses } (i,j) \text{ in } D_f, \\ -1, & \text{if } P \text{ traverses } (j,i) \text{ in } D_f, \\ 0, & \text{otherwise} \end{cases} \qquad \text{for all } (i,j) \in A.$$

We also want to define such a (directed) incidence vector in the same manner for paths that does not necessarily exist in D_f.

Definition 6.17. *For an undirected path $P_{u,v}$ from node u to v in D, we define $\chi(P_{u,v}) \in \{-1,0,1\}^A$ as the* (directed) incidence vector of $P_{u,v}$ *as*

$$\chi_{ij}(P_{u,v}) := \begin{cases} 1, & \textit{if } P_{u,v} \textit{ traverses } (i,j) \textit{ in its forward direction,} \\ -1, & \textit{if } P_{u,v} \textit{ traverses } (i,j) \textit{ in its backward direction,} \\ 0, & \textit{otherwise} \end{cases} \quad \textit{for all } (i,j) \in A.$$

We also define the cost of the path as $c(P_{u,v}) = \sum_{(i,j)\in A} \chi_{ij}(P_{u,v}) c_{ij}$, i.e., the cost of an arc traversed in forward direction contributes positively to the total cost, whereas the cost of a backwards-traversed arc contributes negatively. The incidence vector and the cost of a cycle C in D (with a given orientation) are defined analogously.

These definitions can be seen as also allowing forward and backward arcs in D_f even if the flow value equals the lower or upper capacity, i.e., $A^+ := \{(i,j) : (i,j) \in A,\, f_{ij} \leq u_{ij}\}$ and $A^- := \{(j,i) : (i,j) \in A,\, f_{ij} \geq l_{ij}\}$. Then, any undirected path P_D corresponds to a directed path P (with not necessarily free capacity) in D_f and $\chi(P) = \chi(P_D)$. This leads to the following property.

Property 6.18. *Let P (C) be a directed path (cycle) in D_f and P_D (C_D) its equivalent in D. Then $\chi(P) = \chi(P_D)$ ($\chi(C) = \chi(C_D)$) and $c(f,P) = c(P_D)$ ($c(f,C) = c(C_D)$).*

With these definitions we can prove the following results:

Theorem 6.19. *Each undirected cycle C_D in D can be represented by a composition of the unique induced cycles*

$$C_{ij} \textit{ for all } (i,j) \in C_D \setminus T.$$

Proof. Let C_D be an undirected cycle in D. First, we decompose the cycle C_D into non-tree arcs $a_k = (i_k, j_k)$ and possibly empty tree paths $P^T_{j_l, i_{l+1}}$:

$$C_D = \{a_1, P^T_{j_1,i_2}, a_2, P^T_{j_2,i_3}, \ldots, a_t, P^T_{j_t,i_1}\}$$

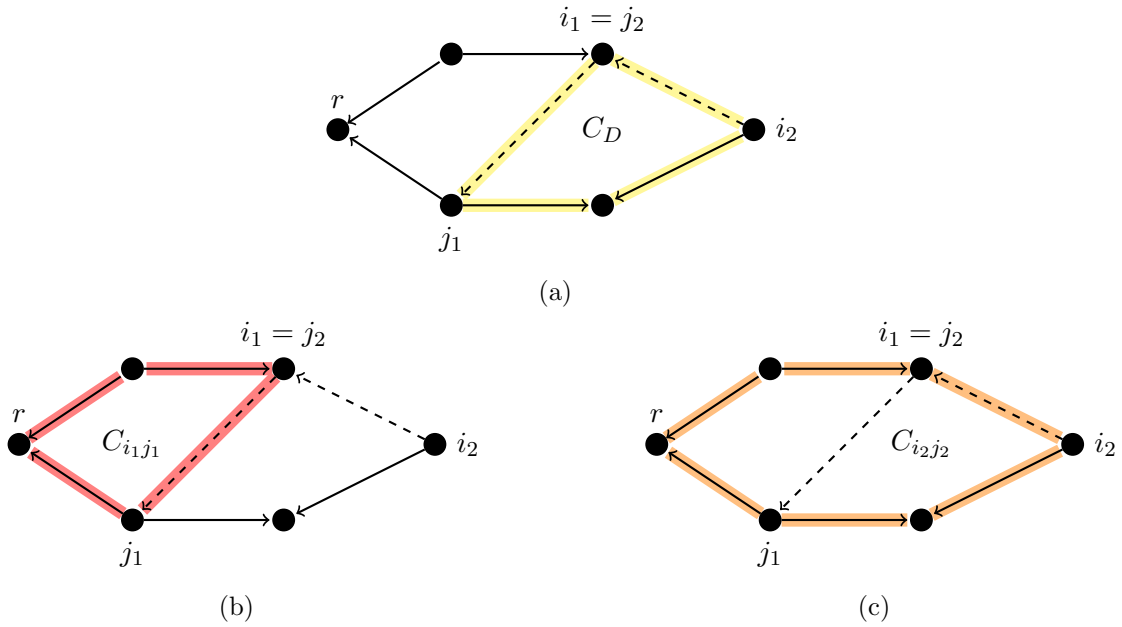


Figure 6.4: (a) A graph with an (undirected) cycle C_D and (b) (c) the two induced cycles whose composition equals C_D. Non-tree arcs are marked as dashed line. It holds $C_D = (i_1, j_1) \circ P^T_{j_1,i_1} \circ (i_2, j_2) \circ P^T_{j_2,i_1} = C_{i_1 j_1} \circ C_{i_2 j_2}$. Here $P^T_{j_2 i_1} = \emptyset$.

Now we consider the composition of the induced cycles C_{a_k}

$$
\begin{aligned}
& C_{a_1} \circ C_{a_2} \circ \cdots \circ C_{a_t} \\
=& a_1 \circ P^T_{j_1,i_1} \circ a_2 \circ P^T_{j_2,i_2} \circ \cdots \circ a_t \circ P^T_{j_t,i_t} && (C_k = a_k \circ P^T_{j_k,i_k}) \\
=& a_1 \circ P^T_{j_1,r} \circ P^T_{r,i_1} \circ a_2 \circ P^T_{j_2,r} \circ P^T_{r,i_2} \circ \cdots \circ a_t \circ P^T_{j_t,r} \circ P^T_{r,i_t} && (P^T_{j_k,i_k} = P^T_{j_k,r} \circ P^T_{r,i_k}) \\
=& a_1 \circ P^T_{j_1,r} \circ P^T_{r,i_2} \circ a_2 \circ P^T_{j_2,r} \circ \cdots \circ P^T_{r,i_t} \circ a_t \circ P^T_{j_t,r} \circ P^T_{r,i_1} && (\circ \text{ is commutative}) \\
=& a_1 \circ P^T_{j_1,i_2} \circ a_2 \circ P^T_{j_2,i_3} \circ \cdots \circ P^T_{j_{t-1},i_t} \circ a_t \circ P^T_{j_t,i_1} && (P^T_{j_k,r} \circ P^T_{r,i_{k+1}} = P^T_{j_k,i_{k+1}}) \\
=& \{a_1, P^T_{j_1,i_2}, a_2, P^T_{j_2,i_3}, \ldots, a_t, P^T_{j_t,i_1}\} && (\text{all components are disjoint}) \\
=& C_D
\end{aligned}
$$

For illustration see Figure 6.4.

□

Any cycle $C \in D_f$ yields an undirected cycle C_D which can be represented by a composition of the unique cycles C_{ij} for all $i, j \in C_D \setminus T$. We can extend this result by showing that the incidence vector and the cost are closed under this composition. For simplicity, we will denote an arc $(i, j) \notin T$ by a in the following.

Lemma 6.20. *For any cycle $C \in D_f$ let C_D be the undirected equivalent cycle in D. Then, it holds that:*

$$\chi(C) = \sum_{a \in C_D \setminus T} \chi(C_a) \qquad \text{and} \qquad c(C, f) = \sum_{a \in C_D \setminus T} c(C_a),$$

where $\chi(C_a) \in \{-1, 0, 1\}^A$ *is defined by*

$$\chi_{ij}(C_a) := \begin{cases} 1, & \text{if } (i,j) \in \overrightarrow{C_a}, \\ -1, & \text{if } (i,j) \in \overleftarrow{C_a}, \\ 0, & \text{otherwise} \end{cases} \qquad \text{for all } (i,j) \in A.$$

Proof. Let C be a cycle in D_f, C_D the corresponding cycle in D and let a be a non-tree arc in $C_D \setminus T$. If $a = (i,j) \in L$, the cycle C_D traverses it in the forward direction, since only (i,j) is present in D_f but (j,i) is not. Similarly, if $a \in U$ then C_D traverses it in its backward direction. So by Definition 2.74 we have $\chi_a(C_D) = \chi_a(C_a) = \sum_{a \in C_D \setminus T} \chi_a(C_a)$. Now we can follow the previous proof. Note that each time the sub-paths canceled each other out by the composition, they could be seen as the same sub-path but traversed in opposite directions. This way, the incidence vectors would sum up to zero for these sub-paths. The tree-paths $C_D \cap T$ are traversed in the correct direction by the composition of cycles C_{a_k} due to their correct orientation.

Formally, let $\chi^a(C_a)$ be the vector having only $\chi_a(C_a)$ in the a-component, i.e., $\chi^a_a(C_a) = \chi_a(C_a)$ and 0 otherwise. So we have $\chi(C_a) = \chi^a(C_a) + \chi(P^T_{j,i})$. Again, suppose that $C_D = \{a_1, P^T_{j_1,i_2}, a_2, P^T_{j_2,i_3}, \ldots, a_t, P^T_{j_t,i_1}\}$. Then

$$\begin{aligned}
&\chi(C_{a_1}) + \chi(C_{a_2}) + \cdots + \chi(C_{a_t}) \\
=&\chi^{a_1}(C_{a_1}) + \chi(P^T_{j_1,i_1}) + \chi^{a_2}(C_{a_2}) + \chi(P^T_{j_2,i_2}) + \cdots + \chi^{a_t}(C_{a_t}) + \chi(P^T_{j_t,i_t}) \\
=&\chi^{a_1}(C_{a_1}) + \chi(P^T_{j_1,r}) + \chi(P^T_{r,i_1}) + \chi^{a_2}(C_{a_2}) + \chi(P^T_{j_2,r}) + \chi(P^T_{r,i_2}) \\
&+ \cdots + \chi^{a_t}(C_{a_t}) + \chi(P^T_{j_t,r}) + \chi(P^T_{r,i_t}) \\
=&\chi^{a_1}(C_{a_1}) + \chi(P^T_{j_1,r}) + \chi(P^T_{r,i_2}) + \chi^{a_2}(C_{a_2}) + \chi(P^T_{j_2,r}) \\
&+ \cdots + \chi(P^T_{r,i_t}) + \chi^{a_t}(C_{a_t}) + \chi(P^T_{j_t,r}) + \chi(P^T_{r,i_1}) \\
=&\chi^{a_1}(C_{a_1}) + \chi(P^T_{j_1,i_2}) + \chi^{a_2}(C_{a_2}) + \chi(P^T_{j_2,i_3}) + \cdots + \chi(P^T_{j_{t-1},i_t}) + \chi^{a_t}(C_{a_t}) + \chi(P^T_{j_t,i_1}) \\
=&\chi^{a_1}(C_D) + \chi(P^T_{j_1,i_2}) + \chi^{a_2}(C_D) + \chi(P^T_{j_2,i_3}) + \cdots + \chi(P^T_{j_{t-1},i_t}) + \chi^{a_t}(C_D) + \chi(P^T_{j_t,i_1}) \\
=&\chi(C_D)
\end{aligned}$$

So we have $\chi(C_D) = \sum_{a \in C_D \setminus T} \chi(C_a)$. To show that the cost is equivalent is now easy:

$$\begin{aligned}
c(C_D) &= \sum_{(i,j) \in A} \chi_{ij}(C_D) c_{ij} = \sum_{(i,j) \in A} \sum_{a \in C_D \setminus T} \chi_{ij}(C_a) c_{ij} \\
&= \sum_{a \in C_D \setminus T} \sum_{(i,j) \in A} \chi_{ij}(C_a) c_{ij} = \sum_{a \in C_D \setminus T} c(C_a)
\end{aligned}$$

□

Now we want to show that we can express any optimal flow by an initial optimal tree solution and a conical combination of certain C_a. Let f be an optimal tree solution. Property 2.66 gives us that any feasible integer flow f' can be written as $f' = f + \sum_{i=1}^k \mu_i \chi(C_i)$

for some $C_i \in D_f$ and integers $\mu_i > 0$. Lemma 6.20 yields

$$f' = f + \sum_{i=1}^{k} \mu_i \chi(C_i) = f + \sum_{i=1}^{k} \mu_i \sum_{a \in C_i \setminus T} \chi(C_a).$$

Rearranging the sums and summarizing the cycles and coefficients yields

$$f' = f + \sum_{a \notin T} \lambda_a \chi(C_a).$$

Here, each cycle C_a is unique and different from the other induced cycles $C_{a'}$ with $a \neq a'$. Because the cycle C_a only consists of tree arcs and the arc $a \notin T$ (Property 2.75), a is not included in any of the other induced cycles $C_{a'}$ with $a \neq a'$. Hence, since f' is a feasible flow, it must hold that $0 \leq \lambda_a \leq u_a$. Otherwise, the flow value f_a of the arc a would not satisfy its lower or upper boundary. This yields a maximum of $(u_a + 1)$ different values for λ_a.

Suppose now that f' is an optimal integer solution, then it holds that $c(f') = c(f)$. Again by Property 2.66 and Lemma 6.20 we can show analogously that

$$c(f') = c(f) + \sum_{a \notin T} \lambda_a c(C_a).$$

Due to the optimality of f it holds that $c(C_a) \geq 0$ and therefore we have $\lambda_a = 0$ for all arcs $a \notin T$ with $c(C_a) > 0$. Let the set S consist of all non-tree arcs with zero reduced cost, i.e.,

$$S = \{a \notin T \,|\, \overline{c}_a = 0\}.$$

Then with Property 2.75, we can reformulate the above equation for any other optimal integer flow f' and an initial optimal tree solution f as

$$f' = f + \sum_{a \in S} \lambda_a \chi(C_a).$$

We formalize the above argumentation in Theorem 6.21.

Theorem 6.21. *Let f be an optimal tree solution with an associated tree structure (T, L, U) and let f^* be another integer flow. Then f^* is optimal if and only if*

$$f^* = f + \sum_{a \in S} \lambda_a \chi(C_a)$$

with

(i) $\lambda_a \in \mathbb{Z}$ *with* $0 \leq f_e + \sum_{a \in S} \lambda_a \chi_e(C_a) \leq u_e \quad \forall\, e \in A$,

(ii) the set S consists of all non-tree arcs with zero reduced cost: $S = \{a \notin T \,|\, \overline{c}_a = 0\}$.

Proof. "⇒": Property 2.66 gives us $f^* = f + \sum_{i=1}^{k} \mu_i \chi(C_i)$ for some C_i in D_f and $\mu_i > 0$. With Property 6.18, we can express f^* through the corresponding cycles $C_{D,i}$ in D:

$$f^* = f + \sum_{i=1}^{k} \mu_i \chi(C_{D,i}) \overset{\text{Lemma 6.20}}{=} f + \sum_{i=1}^{k} \mu_i \sum_{a \in C_{D,i} \setminus T} \chi(C_a)$$

Rearranging the sums and summarizing the cycles and coefficients yield

$$f^* = f + \sum_{a \notin T} \lambda_a \chi(C_a)$$

Since f^* is optimal and thereby in particularly feasible f^* satisfies the capacity constraint of all arcs, which is expressed by (i).

The cost of f^* equals the cost of f and we can show in a similar way like above that $c(f^*) = c(f) + \sum_{a \notin T} \lambda_a c(C_a)$. Therefore the cost of all considered cycles C_a must be zero. Remember that no cycle can have negative cost because of the negative cycle optimality condition. Property 2.75 gives us that $c(C_a) = \overline{c}_a$ or $-\overline{c}_a$. So only arcs $a \in S$ according to (ii) can be considered for representing f^*.

"⇐": Now consider $f^* = f + \sum_{a \in S} \lambda_a \chi(C_a)$ with (i) and (ii) holding. Then f^* satisfies the capacity constraints because of (i). It also satisfies the flow balance constraints because we augmented the flow on cycles, so the flow balance at every node remains. The cost of each cycle that is used for augmentation is zero because of (ii) and Property 2.75. So the cost of f^* is $c(f) + \sum_{a \in S} \lambda_a c(C_a) = c(f)$ and therefore f^* is optimal. □

Moreover, we can show the following:

Lemma 6.22. *Let f be an optimal tree solution, $f^1 = f + \sum_{a \in S} \lambda_a^1 \chi(C_a)$ and $f^2 = f + \sum_{a \in S}^{k} \lambda_a^2 \chi(C_a)$. If $\lambda^1 \neq \lambda^2$ then it holds $f^1 \neq f^2$.*

Proof. Let $a' \in S$ be an arc, such that $\lambda_{a'}^1 \neq \lambda_{a'}^2$. Since a' is contained in the cycle $C_{a'}$ and cannot be contained in any other induced cycle, it follows that $f^1 = f + \sum_{a \in S} \lambda_a^1 \chi(C_a)$ and $f^2 = f + \sum_{a \in S} \lambda_a^2 \chi(C_a)$ have different flow values on the arc a'. So we have $f_{a'}^1 \neq f_{a'}^2$. □

With the above considerations, any optimal integer solution can be obtained from an initial optimal tree solution and a conical combination of incidence vectors of all zero cost induced cycles with bounded integer coefficients. Since any λ_a for an arc $a \in S$ can take at most $(u_a + 1)$ different values this gives the following result.

Theorem 6.23. *Let $|F|$ be the number of all optimal integer flows in D. Then*

$$|F| \leq \max(1, \prod_{a \in S} (u_a + 1)).$$

□

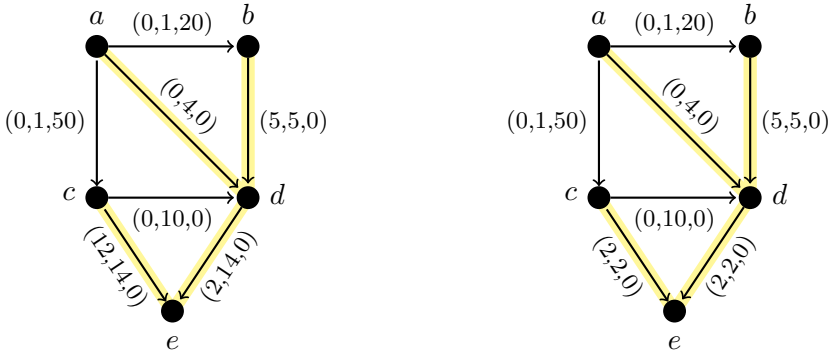


Figure 6.5: Two graphs whose arcs are labeled by $(f_{ij}, u_{ij}, \bar{c}_{ij})$. Arcs in yellow represent the tree arcs.

We also can give a lower bound for $|F|$. Let f be an optimal flow. Considering one induced cycle C_a with $a \in S$, we can generate at least as many new optimal flows as the maximum free capacity of the cycle, denoted with $u(C_a)$ allows. Each unit of flow that is added or removed from the cycle yields another flow. Considering another induced cycle $C_{a'}$ instead of C_a gives $u(C_{a'})$ many different flows. Since we can augment over each induced cycle separately, we get the following lower bound for $|F|$.

Theorem 6.24. *Let $|F|$ be the number of all optimal integer flows in D. Then*

$$\min(1, \sum_{a \in S} u(C_a)) \leq |F|.$$

With the same argumentation, we can give the following bounds for the total number of feasible integer flows, denoted by $|\mathcal{F}|$. Here, we do not have to restrict ourselves to induced cycles with zero reduced cost.

$$\min(1, \sum_{a \notin T} u(C_a)) \leq |\mathcal{F}| \leq \max(1, \prod_{a \notin T} (u_a + 1)).$$

We close this chapter with an example that shows that the bound in Theorem 6.23 can be tight, but is not necessarily so.
Consider Figure 6.5. In (a), the arc (c, d) is the only arc not in T with $c_{ij} = 0$. We can augment ten times over this arc with one flow unit each time. Since there are no more 0-cycles, we get 11 optimal flows, and we have $|F| = \delta := \max(1, \prod_{a \in S}(u_a + 1))$.
In (b), we have the same 0-cycle. However, due to the available capacities and flow values on other arcs, we cannot augment over it. There is only one optimal flow, but $\delta = 10$ and thus $|F| < \delta$. If we increase the arc (c, d) capacities to an arbitrarily large value, we get $|F| \ll \delta$.

6.7 Conclusion

This chapter presented the main theoretical bases and implementation of an algorithm to determine all optimal solutions of a linear integer minimum cost flow problem. For a given optimal solution, the algorithm efficiently determines proper zero cost cycles by using a depth-first search in a reduced network, leading to a new optimal solution. The presented algorithm requires $\mathcal{O}(|F|(m+n)+mn+\zeta)$ time to solve the AOFP. Using the proposed algorithm, a new method is derived to solve the k best flow problem with an improved running time of $\mathcal{O}(kn^3+\zeta)$. Besides, lower and upper bounds for the number of all optimal integer solutions are shown, which is based on the fact that any optimal integer flow can be obtained from an initial optimal tree solution plus a conical combination of all zero cost induced cycles with bounded coefficients.

Future research could focus on structured combinatorial ways to efficiently explore these conical combinations to derive an improved algorithm for determining alternative solutions in network flow problems. An extension of the ideas and techniques presented here to methods for determining supported and nonsupported solutions in BOIMCF and MOIMCF problems will be presented in the following chapters.

7 Output-sensitive Complexity for Multi-Objective Integer Minimum Cost Flow Problems

This chapter addresses the output-sensitive complexity of the MOIMCF problem, providing insights into the time complexity to enumerate all supported efficient solutions and all supported nondominated points. An output-polynomial time algorithm is derived to determine all supported efficient solutions for MOIMCF problems. This is the first approach to solve this general problem in output-polynomial time.

Furthermore, the chapter proves that that an output-polynomial time algorithm for determining all weakly supported nondominated points (or all weakly supported efficient solutions) in a MOIMCF problem with a fixed number of $d \geq 3$ objectives cannot exist unless $\mathbf{P} = \mathbf{NP}$. Additionally, it establishes that there cannot exist an output-polynomial time algorithm that enumerate all supported nondominated points in lexicographic order in the outcome space unless $\mathbf{P} = \mathbf{NP}$.

The results presented in Chapter 7 and Chapter 8 are the outcome of joint work with Michael Stiglmayr. They have resulted in a publication Könen and Stiglmayr [2025a] and a technical report Könen and Stiglmayr [2023], the latter of which has been submitted to the Journal of Combinatorial Optimization.

7.1 Introduction

This chapter focuses on a-posteriori methods, aiming to determine (all) or a suitable subset of the efficient solutions or nondominated points; for a summary of solution concepts in multi-objective optimization, see Serafini [1986].

The subset of interest in this chapter, is the set of all supported efficient solutions. As described in Section 2.3, MOCO problems contain only supported efficient solutions, while the efficient solution set of multi-objective combinatorial optimization problems, such as MOIMCF, also contains nonsupported efficient solutions. The nonsupported efficient solutions typically outnumber the supported ones, where the latter are more straightforward to determine, can serve as high-quality representations [Sayın, 2024], and can be used as a foundation for two-phase methods to generate the entire nondominated point set.

As described in Chapter 4, MOCO problems can have supported, as well as weakly supported solutions. Example 7.1 illustrates an instance of the MOIMCF problem where weakly supported nondominated points exist, and the supported nondominated points form a proper subset of the weakly supported solutions.

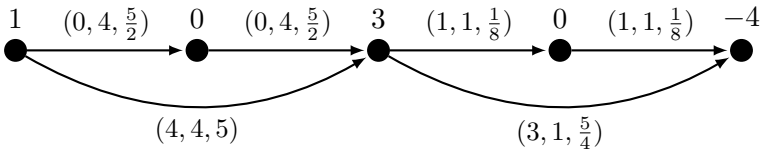


Figure 7.1: The Graph corresponds to a tri-objective MOIMCF problem. Thereby, $l_{ij} = 0$ and $u_{ij} = 4$ for all arcs $(i, j) \in A$. The arcs are labeled with their cost coefficients $(c^1_{ij}, c^2_{ij}, c^3_{ij})$ and the nodes are labeled with their supply/demand b_i.

Example 7.1. *The MOIMCF problem shown in Figure 7.1 has three extreme supported nondominated points* $y^1 = (8, 16, 6)^\top, y^2 = (12, 12, 6)^\top, y^3 = (16, 8, 10)^\top$ *and in addition the following nondominated points* $s^1 = (9, 15, 7)^\top, s^2 = (10, 14, 8)^\top, s^3 = (11, 13, 9)^\top, s^4 = (13, 11, 7)^\top, s^5 = (14, 10, 8)^\top, s^6 = (15, 9, 9)^\top$ *and as well one dominated point* $d^1 = (12, 12, 10)^\top$. *The different points in objective space and their convex hull are illustrated in Figure 7.2. All points* s^i *with* $i \in \{1, \dots, 6\}$ *are weakly supported nondominated points since their preimages are optimal solutions of the corresponding weighted sum problem* P_λ *with* $\lambda = (0.5, 0.5, 0)^\top$. *The points* s^4, s^5, s^6 *are also supported nondominated points since they are optimal solutions of* (P_{λ^2}) *with* $\lambda^2 = (0.25, 0.5, 0.25)^\top$. *However,* s^1, s^2, s^3 *are not optimal for any weighted sum problem* P_λ *with* $\lambda \in \Lambda_d$. *Hence, the set of weakly supported nondominated points is* $\mathcal{Y}_w S = \{y^1, y^2, y^3, s^1, s^2, s^3, s^4, s^5, s^6\}$. *While the set of supported nondominated points is* $\mathcal{Y}_S = \{y^1, y^2, y^3, s^4, s^5, s^6\}$. *Thus, in this example, the set of supported nondominated points is a proper subset of the set of weakly supported nondominated points* $\mathcal{Y}_S \subset \mathcal{Y}_{wS}$. *It is easy to see that there do not exist weights* $\lambda \in \Lambda_d$ *such that* s_1, s_2, *or* s_3 *are optimal solutions for* P_λ *using the* weight space decomposition *which will be formally introduced in Section 7.2.2. Figure 4.2 shows the weight space decomposition of the* upper image *of the MOIMCF instance given in Figure 7.1.*

The clear distinction between the sets of supported nondominated and weakly supported nondominated solutions is also necessary as the corresponding problems differ in their output time complexity. Eusébio and Figueira [2009b] introduced an algorithm that enumerates all supported nondominated points/efficient solutions for MOIMCF problems (assuming extreme supported solutions and corresponding weight vectors are given), based on zero-cost cycles in the incremental graph associated with the corresponding parametric network flow problems. They conclude that their proposed algorithm is the first step in developing further zero-cost cycle algorithms for solving MOIMCF problems. However, they do not provide a specific method for determining those zero-cost cycles. Eusébio and Figueira [2009b] rely on a definition, which is equivalent to what we define as weakly supported nondominated points. However, their proposed algorithm computes only supported efficient solutions whose images lie in the maximally nondominated faces. We refer to Example 7.1.

This chapter will show that there is no *output-polynomial* time algorithm to determine all weakly supported nondominated points (or weaklysupported efficient solutions) for a MOIMCF problem with a fixed number of objectives, unless $\mathbf{P} = \mathbf{NP}$. This is proven

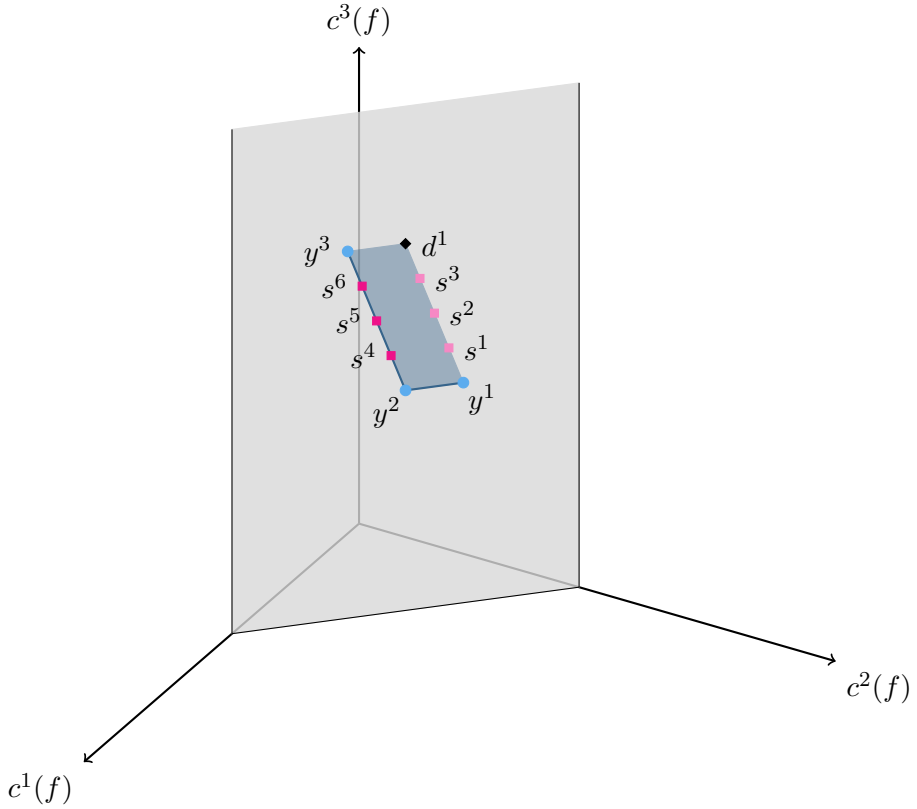


Figure 7.2: The figure shows the convex hull $\text{conv}(\mathcal{Y})$ of the given problem in Figure 7.1 in blue, all its integer vectors and the hyperplane $h = \{y \in \mathcal{Y} \colon 0.5\, y_1 + 0.5\, y_2 = 12\}$ in gray. The red rectangles on the edge between (y^2, y^3) are the vectors s^4, s^5, and s^6. The light red rectangles on the edge between the edge (y^1, d^1) are the vectors s^1, s^2, and s^3, which would be dominated in the non-integer case by the points on the edge (y^1, y^2). The maximally nondominated faces are the edges (y^1, y^2) and (y^2, y^3). Note that s^1,s^2, and s^3 would lie on $\text{conv}(\mathcal{Y})$ but not in any of the maximally nondominated faces. The vectors s^1, s^2, and s^3 would also lie on the boundary of the upper image.

by showing that an output-polynomial algorithm cannot exist for the multi-objective *s-t*-path problem (MOSP) with more than two objectives. Additionally, it shows that there cannot exist an output-polynomial time algorithm for the enumeration of all supported nondominated points that determine the points in an lexicographically ordered way in the outcome space unless $\mathbf{P} = \mathbf{NP}$. In contrast, we derive *output-polynomial enumeration* algorithms to determine all supported efficient solutions, first for the bi-objective case and afterward for every fixed number of objectives.

The approach consists of two phases: First, the algorithm determines all extreme supported nondominated points of the upper image and the associated weight vector for each maximally nondominated face. Hence, the approach successively determines all efficient solutions for each maximally nondominated face by determining all optimal solutions for the linear weighted-sum scalarization (single-objective parametric network flow) problem with the corresponding weight vector using the previously presented Algorithm 9 in Section 6.4 to determine all optimum integer flows in a network. The method successively searches for *proper zero-cost cycles* in linear time by using a modified depth-first search technique.

Given a BOIMCF problem and using the enhanced parametric network approach [Raith and Sedeño-Noda, 2017] to determine all N extreme nondominated points in $\mathcal{O}(N \cdot n(m + n \log n))$ time in a first step, results in an $\mathcal{O}(N \cdot n(m + n \log n) + \mathcal{S}(m + n))$ time algorithm to determine all $\mathcal{S}$ supported efficient solutions to a BOIMCF problem.

Given a MOIMCF problem with a fixed number of d objectives, the dual Benson's algorithm Ehrgott et al. [2012] can be used for the first phase. In addition, while the lexicographic MCFP can be solved in polynomial time Hamacher et al. [2007b], we can use the lexicographic dual Benson's algorithm presented in Bökler and Mutzel [2015], which determines all extreme supported nondominated points, all facets of the *upper image* and the *weight space decomposition* in $\mathcal{O}(N^{\lfloor \frac{d}{2} \rfloor}(\text{poly}(n, m) + N \log N))$. Note that in phase two, the corresponding image of some solutions may lie in more than one maximally nondominated face and, therefore, are included in more than one face. The approach yields an $\mathcal{O}(N^{\lfloor \frac{p}{2} \rfloor}(\text{poly}(n, m) + N \log N + N^{\lfloor \frac{p}{2} \rfloor}\mathcal{S}(m + n + N^{\lfloor \frac{p}{2} \rfloor}p))$ time algorithm to determine all $\mathcal{S}$ supported efficient solutions for a d-objective MOIMCF problem. To our knowledge, this is the first output-polynomial time algorithm to determine the complete set of supported efficient solutions.

The following table summarizes the contribution of the chapter and the existing results from the literature on the existence of output-polynomial time algorithms for the MOIMCF problem w.r.t. the different subsets of interest. A check mark indicates existence, a cross indicates that the existence of such an algorithm can be ruled out unless $\mathbf{P} = \mathbf{NP}$, and the question mark indicates that this problem remains an open question.

Note that an output-polynomial time algorithm for determining all supported efficient solutions is insufficient for the computation of all supported nondominated points in output-polynomial time since there may be exponentially many solutions mapping to the same points. It is easy to check that all extreme supported efficient solutions can be determined in output-polynomial time due to the existence of the output-polynomial time algorithm for the extreme supported nondominated points. Note that the algorithm

Table 7.1: Results on the existence of output-polynomial time algorithms for the MOIMCF problem w.r.t. the different subsets of interest.

	extreme supported	supported	weakly supported	all
nondominated points	✓[1]	?	✗[2]	✗[3]
efficient solutions	✓[4]	✓[5]	✗[2]	✗[6]

[1] Bökler and Mutzel [2015] and Ehrgott et al. [2012]
[2] Transferable from Bökler and Mutzel [2015] and Ehrgott et al. [2012].
[3] Theorem 7.6
[4] Theorem 7.4
[5] Bökler et al. [2017]
[6] Transferable from Bökler et al. [2017], see Lemma 7.5

in Eusébio and Figueira [2009b] would also determine all supported efficient solutions in output-polynomial time when all maximally nondominated faces and corresponding weight vectors are already given.

The remainder of the chapter is structured as follows. Section 7.2 presents the algorithms to determine all supported efficient solutions for BOIMCF problems as well as supported efficient solutions for MOIMCF problems in output-polynomial time. In contrast, Section 7.3 proves that an output-polynomial time algorithm for determining all weakly supported solutions cannot exist unless $\mathbf{P} = \mathbf{NP}$. Furthermore, it establishes that no output-polynomial time algorithm for the enumeration of all supported nondominated points that determine the vectors in an lexicographically ordered way in the outcome space can exist unless $\mathbf{P} = \mathbf{NP}$.

7.2 An Output-Polynomial Time Algorithm to Determine all Supported Efficient Solutions

In this section, output-polynomial time algorithms are derived that determine all supported efficient flows for MOIMCF. The algorithms are based on the determination of all optimal flows for a sequence of single-objective parametric network flow problems, each corresponding to a maximally nondominated face. The approach consists of two phases and relies on the following widely known fact for any integer supported flow, see, e.g., Gal [1977].

Theorem 7.2. *A flow f is contained in $F_{\mathcal{X}}$, i. e., its image of $c(f)$ lies on a maximally nondominated face $F_{\mathcal{Y}}$ w. r. t. an associated weight vector $\lambda \in \Lambda_p$, if f is an optimal solution to the parametric network flow program (weighted-sum scalarization) (P_λ).*

Any image of a supported flow must lie in at least one maximally nondominated face, and any integer point in a maximally nondominated face corresponds to a supported integer

flow. Assuming that for each maximally nondominated face $F_i \in \{F_1, \dots, F_t\}$ one optimal solution f^i and the corresponding weighting vectors λ^i are given, the problem of determining all supported flows reduces to determining all optimal flows for each parametric single-objective problem (P_{λ^i}). These optimal solutions can be determined by using Algorithm 9 for determining all optimum flows for single-objective minimum cost flow problems presented in Section 6.4, which we refer to as the all optimum flow (AOF) algorithm. As described in Section 6.4, the AOF algorithm successively searches for so-called *proper zero-cost* cycles efficiently by using a modified depth-first search technique. By Theorem 6.11 it follows that given an initial optimal integer flow f, we can determine all optimal integer flows in $\mathcal{O}(|F|(m+n)+m\,n)$ time for a single-objective minimum cost flow problem, where F is the set of all optimal integer flows. Therefore, we divide the approach into two phases: In phase one, we determine all extreme supported nondominated points and one weighting vector for each maximally nondominated face. In phase two, we apply the AOF algorithm to the corresponding weighted-sum program for each maximally nondominated face.

The extreme nondominated points and the weighting vectors for each maximally nondominated face can be determined much easier in the case of two objectives. Therefore, we start by deriving an algorithm for BOIMCF problems and consider the general case of MOIMCF problems afterward.

7.2.1 Bi-Objective Minimum Cost Flow Problem

In the bi-objective case, the set of supported flows is equal to the set of weakly supported flows since every weakly nondominated face contains exactly one nondominated point (namely an extreme supported nondominated point), which dominates the complete face.

First, we determine all $|\mathcal{Y}_{ES}|$ extreme supported nondominated points and precisely one corresponding extreme supported flow by using the enhanced parametric programming approach in $\mathcal{O}(\zeta+|\mathcal{Y}_{ES}|n(m+n\log n))$ time given in Raith and Sedeño-Noda [2017], where ζ denotes the time required to solve a given single-objective minimum cost flow problem. Also, the algorithm stores one extreme flow for each extreme supported nondominated point.

In the bi-objective case, every maximally nondominated face $F_{\mathcal{Y}}$ of $\text{conv}(\mathcal{Y})$ is a line segment connecting two adjacent extreme supported nondominated points if there is more than one nondominated point ($|\mathcal{Y}_N| > 1$). A maximally nondominated face can only have dimension zero if there is only one extreme supported nondominated point, which implies that there is only one nondominated point (or, in other words, the Ideal point is feasible).

In the following, we will derive a procedure to determine the complete set of all supported efficient flows. For that, we will determine all supported flows whose images lie on the maximally nondominated edges. Let $y^1, \dots, y^{|\mathcal{Y}_{ES}|}$ be the extreme supported nondominated points obtained by the enhanced parametric programming approach [Raith and Sedeño-Noda, 2017] and let $f^1, \dots, f^{|\mathcal{Y}_{ES}|}$ be a set of corresponding extreme supported flows each mapping to one extreme supported point. Moreover, we sort the set of extreme

supported nondominated points and flows $\{y^i = (c^1(f^i), c^2(f^i)), f^i \colon i \in \{1, \dots, |\mathcal{Y}_{ES}|\}\}$ by non-decreasing values of c^1. For each pair of consecutive extreme points y^i and y^{i+1}, we determine the weighting vector $\lambda^i \in \Lambda_2$ that corresponds to the normal of the maximally nondominated facet F_i connecting the extreme points y^i and y^{i+1}:

$$\lambda^i := \begin{pmatrix} c^2(f^i) - c^2(f^{i+1}) \\ c^1(f^{i+1}) - c^1(f^i) \end{pmatrix}$$

Then f^i and f^{i+1} are both optimal flows for the single-objective weighted-sum (MCF) program (P_{λ^i}) [Eusébio and Figueira, 2009a]. Hence, determining all optimal solutions for (P_{λ^i}) gives all supported efficient flows whose image lies in between F_i. Figure 7.3 illustrates the objective function of the weighted-sum problem (P_{λ^i}) and the maximally nondominated face between two consecutive extreme points in the outcome space.

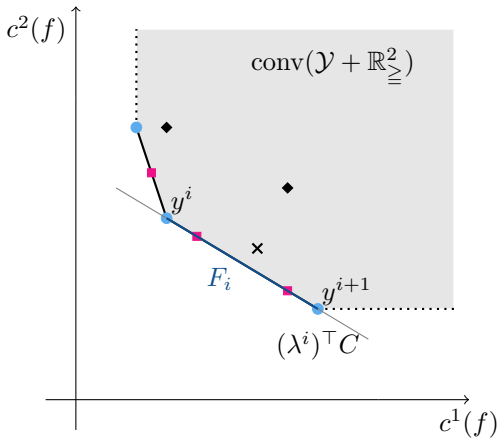


Figure 7.3: Illustration of two neighboring extreme points y^i and y^{i+1}, the maximally nondominated edge F_i in blue and the cost vector of $(\lambda^i)^\top C$.

Theorem 7.3. *Given the directed network $(D, l, u, b, (c^1, c^2)^\top)$, Algorithm 12 determines the set of all supported flows $\mathcal{X}_S$ in $\mathcal{O}(|\mathcal{Y}_{ES}|n(m + n \log n) + |\mathcal{X}_S|(m + n))$ time.*

Proof. Any supported nondominated point must lie at least on one maximally nondominated edge, and only extreme supported nondominated points y^i for $i = 2, \dots, |\mathcal{Y}_{ES}| - 1$ lie in two maximally nondominated edges, namely F_i and F_{i+1}. According to Theorem 7.2, all supported flows can be determined as optimal flows for the different weighted-sum problems (P_{λ^i}). Moreover, nonsupported flows correspond to suboptimal solutions for all weighted-sum problems. Since we only store flows for $i = 2, \dots, |\mathcal{Y}_{ES}| - 1$ where $c^1(f) \neq c^1(f^i)$, no supported flow is stored twice. Hence, Algorithm 12 determines the complete set of all supported flows.

The enhanced parametric network approach in Raith and Sedeño-Noda [2017] requires $\mathcal{O}(|\mathcal{Y}_{ES}|n(m + n \log n) + \zeta)$ time, where ζ is the time required to solve a single-objective minimum cost flow problem. Since the algorithm determines the extreme points in a

Algorithm 12: FindAllSupportedEfficientFlowsBiObjective

Data: (D, l, u, b, c^1, c^2)
Result: The set of all supported flows $\mathcal{X}_S$

```
1 X_S = ∅
  // Determine all |Y_ES| extreme supported nondominated points y^i and for
     each one corresponding extreme flow f^i, sorted non-decreasingly in
     c^1(f)
2 {(y^i, f^i): i ∈ {1,...,|Y_ES|}} = EnhancedParametricNetworkAlgortihm(D)
3 for i = 1,...,|Y_ES| − 1 do
4     λ_1^i = c^2(f^i) − c^2(f^{i+1});   λ_2^i = c^1(f^{i+1}) − c^1(f^i)
5     c̄ ← Determine reduced costs
6     X_S = X_S ∪ FindAllOptimalFlows(P_{λ^i}, f^i)
      // Return only flows with c^1(f) ≠ c^1(f^i) to avoid repetitions.
7 return X_S.
```

decreasing order of $c^1(f)$, we do not need additional time to sort the extreme points. Defining the weight vectors λ^i for $i = 1, \dots, |\mathcal{Y}_{ES}| - 1$ and building the network with the corresponding cost function takes $\mathcal{O}(|\mathcal{Y}_{ES}|(n+m))$ time. Determining all $\mathcal{F}_i$ optimal flows for one weighted-sum problem (P_{λ^i}) using the Algorithm 9, requires $\mathcal{O}(\mathcal{F}_i(m+n) + m\,n)$ time, where f^i is a corresponding optimal solution to F_i. Since the image of every supported efficient solution lies at most on two maximally nondominated faces, it holds that $\sum_{i=1}^{|\mathcal{Y}_{ES}|-1} \mathcal{F}_i < 2\,\mathcal{S}$. We must consider $|\mathcal{Y}_E| - 1$ of these single-objective minimum cost flow problems corresponding to the maximally nondominated faces. Hence, Algorithm 12 requires overall $\mathcal{O}(|\mathcal{Y}_{ES}|n(m + n\log n) + |\mathcal{X}_S|(m+n))$ time. □

Note that the determination of all supported efficient flows could easily be integrated during the enhanced parametric network approach [Raith and Sedeño-Noda, 2017]. Whenever a new extreme supported nondominated point is found, determine all optimal flows to (P_{λ^i}) with the AOF algorithm.

7.2.2 Multi-Objective Minimum Cost Flow Problems

In the following, we derive an algorithm to determine the complete set of all supported efficient flows, and hence, all supported nondominated points for the multi-objective integer minimum cost flow problem.

First, we need to determine the set of extreme supported nondominated points and the associated weight space decomposition. We can determine all extreme supported nondominated points using the dual Benson's algorithm [Ehrgott et al., 2012]. However, since the lexicographic version of a MOIMCF problem can be solved in polynomial time [Hamacher et al., 2007b], we may also use the lexicographic dual Benson's algorithm recently presented in Bökler and Mutzel [2015]. Both versions work with the upper image $\mathcal{Y}^{\geqq} := \operatorname{conv}(\mathcal{Y} + \mathbb{R}^d_{\geq})$ and its dual polyhedron, or *lower image* $\mathcal{D}$.

While we work with normalized weight vectors $\lambda \in \Lambda_p$, it suffices to consider the so-called *projected weight space* $\{(\lambda_1, \dots, \lambda_{p-1}) \in \Lambda_{p-1} \colon \sum_{i=1}^{p-1} \lambda_i < 1\}$ and calculate the normalized weighting vector $\ell(v) \coloneqq (v_1, \dots, v_{p-1}, \sum_{i=1}^{p-1} v_i)$ of a projected weight v when needed. The dual problem (D_λ) of the weighted sum scalarization (P_λ) is given by

$$\begin{aligned} \max \quad & b^\top u \\ \text{s.t.} \quad & A^\top u = C^\top \lambda \\ & u \in \mathbb{R}^m_{\geqq}. \end{aligned} \tag{D_λ}$$

The dual polyhedron $\mathcal{D}$ then consists of vectors $(\lambda_1, \dots, \lambda_{p-1}, b^\top u)$ with $\lambda \in \Lambda_p^0$ and solutions u of (D_λ). Following the duality theory of polyhedra, there exists a bijective mapping Ψ between the set of all faces of $\mathcal{Y}^{\geqq}$ and the set of all faces of $\mathcal{D}$ such that Ψ is *order reversing*, i. e., if two faces F_1 and F_2 of $\mathcal{Y}^{\geqq}$ satisfy $F_1 \subseteq F_2$ then $\Psi(F_1) \supseteq \Psi(F_2)$ and $\Psi(F_1)$ and $\Psi(F_2)$ are faces of $\mathcal{D}$, see e.g., Schulze et al. [2019]. Thus, an supported solution of $\mathcal{D}$ corresponds to a facet of $\mathcal{Y}^{\geqq}$, and an supported solution of $\mathcal{Y}^{\geqq}$ corresponds to a facet of $\mathcal{D}$. The dual Benson's algorithm solves a MOLP by computing the extreme points of $\mathcal{D}$. For more details on the dual Benson's algorithm or its lexicographic version, we refer to Bökler and Mutzel [2015] and Ehrgott et al. [2012].

Thus, we obtain all extreme supported nondominated points and one corresponding extreme efficient solution for each of the extreme supported nondominated points, as well as all facets of $\mathcal{Y}^{\geqq}$. On this basis, we yield the *weight space decomposition* using the dual (lexicographic) Benson's algorithm. The set of weighting vectors associated with a point $y \in \mathcal{Y}$ is given by

$$\mathcal{W}(y) \coloneqq \left\{ w \in \Lambda_p^0 \colon w^\top y \leq w^\top y' \text{ for all } y' \in \mathcal{Y}^{\geqq} \right\}.$$

Note that the facets of $\mathcal{Y}^{\geqq}$ may only be *weakly* nondominated, i. e., they might contain dominated (integer feasible) points. Recall that all supported nondominated points can be determined by a parametric MCF problem (P_λ) for some weight vector $\lambda \in \Lambda_p$. However, the weight vectors corresponding to the facets of $\mathcal{Y}^{\geqq}$ might have components equal to zero $\lambda_i = 0$ for $i \in \{1, \dots, p\}$, i. e., $\lambda \in \Lambda_p^0$. In the following, we describe a recursive algorithm to obtain the weight vectors for all maximally nondominated faces.

Let U be the set of all extreme points in the lower image $\mathcal{D}$ and $\{\lambda^u \colon u \in U\}$ the set of corresponding weight vectors. Let, furthermore, be F_u the facet of $\mathcal{Y}^{\geqq}$ corresponding to $u \in U$. Then, we call two extreme points u and u' of $\mathcal{D}$ *adjacent*, if $\dim(F_u \cap F_{u'}) = p - 2$. In the following, we will denote the set of adjacent extreme points for $u \in U$ by $Q_u \subseteq U$.

Recall that the intersection of k adjacent facets yield a $p-k$ dimensional face. For each $\lambda^u \in \Lambda_p$ (i. e., $\lambda^u > 0$) we know that all points on the facet F_u are supported nondominated points. Thus, we only have to solve the all-optimum flow problem on (P_{λ^u}). Since some solutions may lie in the same sub-faces of adjacent facets, we have to ensure that no solution is stored twice. In order to do so, we keep track of the neighboring extreme points during Benson's algorithm and store all already processed adjacent extreme points of $u \in U$ in a list δ_u.

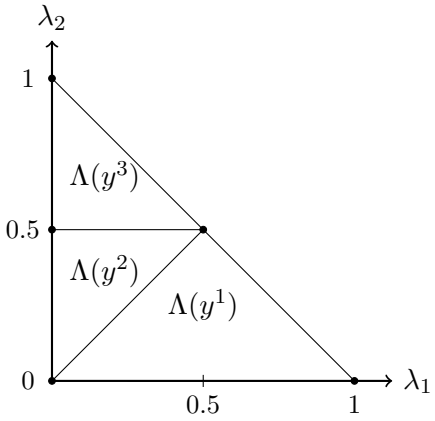


Figure 7.4: Weight space decomposition to the upper image of Figure 7.2. Here it holds $\lambda_3 = 1 - \lambda_2 - \lambda_1$.

There may exist maximally nondominated faces (with dimension less than $p-1$), which are intersections of a number of facets for which the corresponding weight vector equals zero in at least one component. We call these facets *weakly nondominated facets* of $\mathcal{Y}_N^{\geqq}$. In order to determine all supported efficient solutions, we investigate nondominated faces which are intersections of weakly supported facets. With $U_> := \{u \in U : \lambda^u > 0\}$ we denote the set of extreme points of $\mathcal{D}$ corresponding to nondominated facets and with $U_0 := \{u \in U : \lambda^u \geq 0\}$ the set of extreme points corresponding to all weakly nondominated facets ($U_> \subseteq U_0$). Note that weakly nondominated faces can contain supported nondominated points only at its (relative) boundary, while nonsupported nondominated points can be located also in its (relative) interior.

Figure 7.4 presents the weight space decomposition for the example given in Example 7.1 and illustrated in Figure 7.2. Any point in the weight space decomposition corresponds to an λ^u for each $u \in U$. Here $U_> = \varnothing$. However, there do exist weights in the lines in the interior connecting two adjacent extreme points of $\mathcal{D}$ which correspond to the maximally nondominated faces $[y^1, y^2]$ and $[y^2, y^3]$.

Theorem 7.4. *Given the directed network (D, l, u, b, c), Algorithm 13 determines the complete set of all supported flows in $\mathcal{O}(|\mathcal{Y}_{ES}|^{\lfloor \frac{p}{2} \rfloor}(\text{poly}(n,m)+|\mathcal{Y}_{ES}|\log|\mathcal{Y}_{ES}|+|\mathcal{Y}_{ES}|^{\lfloor \frac{p}{2} \rfloor}|\mathcal{X}_S|(m+n+|\mathcal{Y}_{ES}|^{\lfloor \frac{p}{2} \rfloor}p))$ time.*

Proof. Correctness: Any supported efficient flow must lie in at least one maximally efficient face. However, a supported efficient flow can lie in multiple face. Due to Theorem 7.2, all supported flows are found by determining all optimal flows for each weight vector λ^i corresponding to a maximally nondominated face. Moreover, no weakly supported flow can be optimal for a parametric network flow problem with one of these cost functions. While the algorithm iterates through all maximally nondominated faces, only flows are stored that have not been considered yet. Thus, Algorithm 13 determines the complete set of all supported efficient solutions.

Run-time: Benson's Algorithm requires $\mathcal{O}(|\mathcal{Y}_{ES}|^{\lfloor \frac{p}{2} \rfloor}(\text{poly}(n,m) + |\mathcal{Y}_{ES}|\log|\mathcal{Y}_{ES}|))$ time

Algorithm 13: FindAllSupportedEfficientFlows

Data: (D, l, u, b, c)
Result: The complete set of all supported efficient flows

$\{U, Q_u, F_u, \lambda_u, f^u : u \in U\} = \text{BensonLex}(D, l, u, b, c)$
// Determine all extreme points of $\mathcal{D}$, the corresponding facets of $\mathcal{P}$, and weight vectors.
$\delta_u \leftarrow \varnothing \quad \forall u \in U$
for $u \in U_>$ **do**
  $\mathcal{X}_S \leftarrow \mathcal{X}_S \cup \text{FindAllOptimalFlows}(P_{\lambda^u}, f^u)$
  // In FindAllOptimalFlows (Algorithm 9) only store flows f for which $\langle \lambda^u, C\, f\rangle \neq \min\{\langle \lambda^{u'}, C\, f'\rangle : f' \in \mathcal{X}\}$ for any $u' \in \delta_u$
  for $u' \in Q_u$ **do**
    $\delta_{u'} \leftarrow \delta_{u'} \cup \{u\}$
$w_u \leftarrow \{\lambda^{u'} : u' \in \delta_u\} \quad \forall u \in U$
$B \leftarrow \varnothing$
for $u \in U_0 \backslash U_>$ **do**
  $\tilde{U} \leftarrow \{u\}$
  $\mathcal{X}_S \leftarrow \mathcal{X}_S \cup \text{ConsiderSubFaces}(\tilde{U}, U_0 \backslash U_>, B, Q_u, \delta_u, \lambda^u, f^u, w_u \ \forall u \in U)$
  $B \leftarrow B \cup \{u\}$
return $\mathcal{X}_S$

[Bökler and Mutzel, 2015]. Thereafter, we consider each face at most once. The number of all faces can be bounded by $\mathcal{O}(|\mathcal{Y}_{ES}|^{\lfloor \frac{p}{2} \rfloor})$ [Bökler and Mutzel, 2015]. We check for each weakly nondominated face if a strict convex combination with adjacent weight vectors yields a weight vector $\lambda > 0$ component-wise strictly greater than zero. In this case, we call Algorithm 14. Note that λ is not part of the input of Algorithm 13. However, it can be shown that these encoding lengths can be bounded by $\mathcal{O}(\text{poly}(n, m))$ [Bökler and Mutzel, 2015]. The convex combination can be computed in $\mathcal{O}(|\mathcal{Y}_{ES}|^{\lfloor \frac{p}{2} \rfloor} p)$. First, we must create the weight vector through a strictly convex combination for each maximally nondominated face, which is not a facet. Afterwards, we solve the AOF problem for P_λ for all of these maximally nondominated faces in time $\mathcal{O}(F_i(m + n + |\mathcal{Y}_{ES}|^{\lfloor \frac{p}{2} \rfloor} p)) + m\, n)$, where F_i is the number of optimal solutions for the current weighted sum problem P_λ. Additionally, it takes $\mathcal{O}(|\mathcal{Y}_{ES}|^{\lfloor \frac{p}{2} \rfloor} p)$ time to check if the flow is also optimal for an adjacent already considered maximally nondominated face. Since each flow may be contained in all faces, we obtain the bound $\sum F_i \leq \mathcal{O}(|\mathcal{Y}_{ES}|^{\lfloor \frac{p}{2} \rfloor} |\mathcal{X}_S|)$. Hence, Algorithm 13 requires overall $\mathcal{O}(|\mathcal{Y}_{ES}|^{\lfloor \frac{p}{2} \rfloor}(\text{poly}(n, m) + |\mathcal{Y}_{ES}| \log |\mathcal{Y}_{ES}| + |\mathcal{Y}_{ES}|^{\lfloor \frac{p}{2} \rfloor} |\mathcal{X}_S|(m + n + |\mathcal{Y}_{ES}|^{\lfloor \frac{p}{2} \rfloor} p))$ time. □

Algorithm 14: ConsiderSubFaces

Data: $(\tilde{U}, U_0 \backslash U_>, B, Q_u, \delta_u, \lambda^u, f^u, w_u) \quad \forall u \in U$
Result: The set $\bar{\mathcal{X}}$ of all supported efficient flows in the maximally nondominated sub-faces of the facet F_u of $\mathcal{P}$, which are not lying in an already investigated faces

Let $\bar{U} \leftarrow \{u' \in (\bigcap_{u \in \tilde{U}} Q_u \cap U_0 \backslash U_>) \backslash (B \cup \tilde{U})\}$
$\bar{\mathcal{X}} \leftarrow \emptyset$
for $u \in \bar{U}$ **do**
  $\tilde{U} \leftarrow \tilde{U} \cup \{u\}$
  if $|\tilde{U}| \leq p - 1$ **then**
    $\lambda^k \leftarrow \sum_{i=1}^{|\tilde{U}|} \ell_i \lambda^{\tilde{u}_i}$ for an
     $\ell \in \{\sum_{i=1}^{|\tilde{U}|} \ell_i = 1, \ell_i > 0 \quad \forall i \in \{1, \ldots, |\tilde{U}|\}\}, \tilde{u}_i \in \tilde{U}$
    if $\lambda^k > 0$ **then**
      $\bar{\mathcal{X}} \leftarrow \bar{\mathcal{X}} \cup$ FindAllOptimalFlows$(P_{\lambda^k}, f^{\tilde{u}_1})$
      // Only store flows f for which
        $\langle \lambda^k, Cf \rangle \neq \{\min\{\langle \lambda^{u'}, Cf' \rangle \colon f' \in \mathcal{X}\} \colon \lambda^{u'} \in \bigcup_{u \in \tilde{U}} w_u\}$
      for $u' \in \cup_{u \in \tilde{U}} Q_u$ **do**
        $w_{u'} \leftarrow w_{u'} \cup \{\lambda^k\}$
    else
      $\bar{\mathcal{X}} \leftarrow \bar{\mathcal{X}} \cup$ ConsiderSubFaces$(\tilde{U}, U_0 \backslash U_>, B, Q_u, \delta_u, \lambda^u, f^u, w_u \; \forall u \in U)$
  $B \leftarrow B \cup \{u\}$
return $\bar{\mathcal{X}}$

7.3 Output-Sensitive Analysis For Supported Nondominated Points

Section 7.2 presents an *output-polynomial* time algorithm for determining all efficient supported flows for MOIMCF. Unfortunately, this approach is insufficient for computing all supported nondominated points in output-polynomial time, as a single point may correspond to an exponential number of flows.

Analyzing the output-sensitive complexity of specific problems has gained importance in recent years. Several combinatorial problems have been studied, e.g., multi-objective shortest path [Bökler et al., 2017], multi-objective spanning tree problems [Bökler et al., 2017; Okamoto and Uno, 2011], general multi-objective combinatorial optimization problems, and multi-objective linear programs [Bökler and Mutzel, 2015].

The purpose of this section is to give insights into the time complexity for the enumeration of all supported nondominated points for MOIMCF. The section shows that no output-polynomial algorithm to determine all weakly supported solutions can exist, unless **P** = **NP**. Furthermore, it establishes that there cannot exist an output-polynomial time

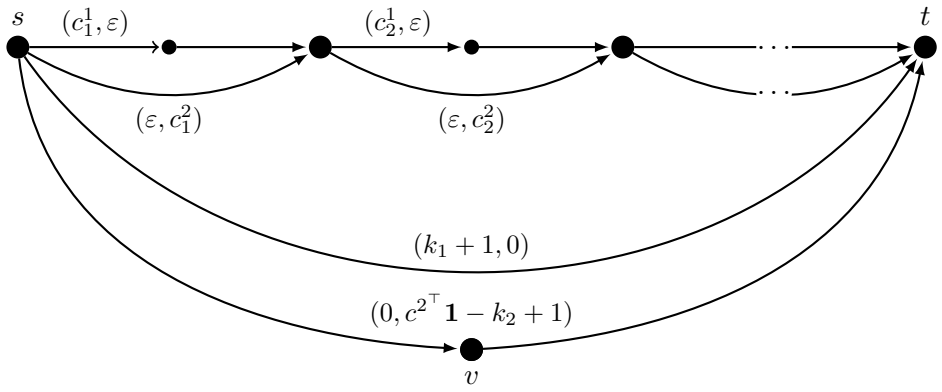


Figure 7.5: Showing the reduction in the proof of Theorem 2.62 with modified costs in order to proof Lemma 7.5. Here $\varepsilon = 1/(n+1)$ and arcs with no label have cost $\mathbf{0}$. Determining another efficient s-t path to the two with cost equal to $(k_1 + 1, 0)$ and $(0, c^{2^\top}\mathbf{1} - k_2 + 1)$ would be an instance of the (KP) problem.

algorithm for the enumeration of all supported nondominated points that determine the points in an lexicographically ordered way in the outcome space unless $\mathbf{P} = \mathbf{NP}$.

7.3.1 Weakly Supported Nondominated Points

As shown in Theorem 4.9, the determination of all weakly supported nondominated points for a MOCO problem with $p+1$ objectives is as hard as the determination of all nondominated points for a MOCO problem with p objectives.

It is well known that the multi-objective shortest path problem (MOSP), a special case of the minimum cost flow problem, is intractable (see Theorem 2.61) and there could not be an output-polynomial algorithm to determine all nondominated points, even in the bi-objective case (see Theorem 2.62).

The determination of all nondominated points of a MOSP, as well as the determination of all efficient solutions of a MOSP can be formulated as an enumeration problem [Bökler et al., 2017]. Again, we denote the finished decision problem for the determination of all nondominated points of a MOSP as $\text{MOSP}_{\mathcal{Y}}^{\text{FIN}}$ and the determination of all efficient solutions of a MOSP as $\text{MOSP}_{\mathcal{X}}^{\text{Fin}}$, respectively.

The same reduction that is used in the proof of Theorem 2.62 given in Bökler et al. [2017] can be extended to show that the $\text{MOSP}_{\mathcal{X}}^{\text{FIN}}$ is also **co-NP**-hard and thus there cannot exist an output-polynomial time algorithm to determine all efficient solutions for the MOSP. However, we have to adjust the costs of some weights. In Figure 7.5, an example of the reduction is given, similar to the one used in the proof of Theorem 2.62 to show the following Lemma. For the sake of simplicity, we do not give a formal proof here. For more details, we refer to the proof of Theorem 2.62 or Bökler et al. [2017].

Lemma 7.5. *There is no output-polynomial time algorithm to determine all efficient solutions for the MOSP problem unless* $\mathbf{P} = \mathbf{NP}$.

The multi-objective shortest path problem (MOSP) is a special case of the multi-objective minimum cost flow problem Ahuja et al. [1993]. The s-t path problem can be transformed into a minimum cost flow problem by setting all arc capacities to one and sending one unit of flow from s to t, with $b_s = 1$, $b_t = -1$, and $b_v = 0$ for all $v \in V \backslash \{s, t\}$. In this modified network, an optimal flow corresponds to the shortest path from s to t. Consequently, the finished or canonical decision problem of the multi-objective minimum cost flow problem is **NP**-hard, and the enumeration problem is intractable even in the case of two objectives. Showing that another nondominated points or efficient flow exists would solve the complement of the knapsack problem.

Thus, due to Theorem 2.62, Lemma 7.5 and Theorem 4.9, it holds that there is no output-polynomial algorithm to determine all weakly supported nondominated points or weakly supported efficient solutions for the MOIMCF problem with a fixed number of $p \geq 3$ objectives.

Theorem 7.6. *Unless* $\mathbf{P} = \mathbf{NP}$*, there is no output-polynomial algorithm to determine all* weakly *supported nondominated points (or weakly supported efficient solutions) for the MOIMCF problem with a fixed number of* $p \geq 3$ *objectives.*

7.3.2 Supported Nondominated Points

To analyze the time complexity of enumerating all supported nondominated points for MOIMCF problems, we examine the case of BOIMCF problems. As a preliminary step, we introduce new problems in single-objective integer MCF, which we abbreviate as MCIF.

Definition 7.7. *Given a single-objective* Minimum Cost Integer Flow Problem *(MCIF) and an integer* $k \in \mathbb{Z}$*. Then the* exact flow problem (EF) *asks whether there exists a flow* f *with cost* $c(f) = k$*.*

We prove that this decision problem is **NP**-complete by reducing it to the well-known **NP**-complete *subset sum* problem (SSP), as introduced in Definition 2.4. For clarity and convenience, we repeat the definition here.

Given a set $N = \{1, \ldots, n\}$ of n items with positive integer weights $w_1, \ldots, w_n$ and a real value k, the *subset sum problem* is to find a subset of N such that the corresponding total weight is exactly equal to k. The formal definition is given by

$$\sum_{j=1}^{n} w_j \, x_j = k$$
$$x_j \in \{0, 1\} \quad \forall j \in \{1, \ldots, n\}$$

Theorem 7.8. *The exact flow problem is* **NP***-complete.*

Proof. Take an instance of the subset sum problem. Create the following instance of an exact flow problem. Create a node i for each $i \in N$ and an additional node $n + 1$. Create two arcs $(i, i + 1)$ $\forall i \in \{1, \ldots, n\}$, one with cost equal to zero and one with cost equal to w_i. All arcs get lower and upper capacities equal to zero and one, respectively. We define

the nodes $1 = s$ and $n + 1 = t$. We set $b_s = -1, b_t = 1$ and $b_i = 0$ for all other nodes. A construction of this instance is illustrated in Figure 7.6. Note that artificial nodes and arcs could be added to prevent multi-arcs.

If we can decide whether a flow $f \in \mathcal{X}$ with $c(f) = k$ exists in polynomial time, we would also be able to solve the subset sum problem in polynomial time. Assume the subset sum problem is solvable, i.e., $\sum_{j=1}^{n} w_j x_j = k$. For each $x_i = 1$ we take the arc $(i, i + 1)$ with cost $c_{i,i+1} = w_i$. For each $x_i = 0$, we take the arcs $(i, i+1)$ with a cost equal to zero. This is a feasible s-t path (feasible flow) with a cost equal to k. On the other side, considering a path P (flow) with cost equal to k, we set $x_i = 1$ if $(i, i + 1) \in P$ that have cost equal w_i and $x_i = 0$ if not. Then x solves the subset problem with the same argumentation as above.

□

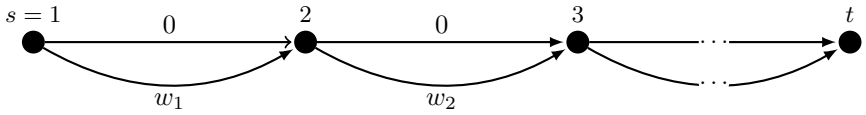


Figure 7.6: The instance of the exact flow problem corresponding to a given subset sum instance.

Theorem 7.8 implies that it is also **NP**-complete to determine if a flow f exists with $k_1 < c(f) < k_2$ with $k_1, k_2 \in \mathbb{Z}$.

In addition, we can prove the following statement.

Theorem 7.9. *The problems to determine the k-th best flow or the k-th smallest distinct cost of a flow are* **NP***-hard.*

Proof. Let the time of finding the k-th best or k-th smallest distinct cost of a flow be $T(n)$. Consider the instance of the exact flow problem for a given subset sum problem as in the proof above. Deciding whether a flow f exists with $c(f) = l$ would solve the subset sum problem. At most, 2^n different flows (item i could be selected or not) could exist. Thus, a binary search for a given flow value is $\mathcal{O}(n)$. It can be concluded that the complexity of the subset sum problem, known to be **NP**-complete, is $\mathcal{O}(nT(n))$. Therefore, the problem of determining the k-th best or k-th smallest distinct cost of a flow is **NP**-hard. □

Next, we will prove that we can determine all distinct costs of the flows for a single-objective minimum cost flow problem in output-polynomial time if we can determine all supported nondominated points for a bi-objective integer minimum cost flow problem in output-polynomial time and vice versa.

Definition 7.10. *Given a single-objective MCIF, the* all distinct cost value flow problem (ACVF) *determines a minimal set of flows that includes all existing different cost values of all feasible flows. In other words, it identifies k different flows $f_1, \ldots, f_k$ such that*

$$c\left(f^1\right) < \ldots < c\left(f^k\right)$$

and there exists no flow f^p such that $c(f^p) \notin \{c(f^1), \ldots, c(f^k)\}$.

Theorem 7.11. *If we can solve the ACVF in output-polynomial time, then we can determine all supported nondominated points of a BOIMCF in output-polynomial time.*

Proof. Each supported nondominated point lies on a maximal nondominated facet of the upper image since its preimage is an optimal solution to a weighted sum scalarization P_λ for a $\lambda \in \Lambda_p$ (i.e., weights are strictly positive). Note that in the bi-objective case, every maximally nondominated face $F_\mathcal{Y}$ of $\text{conv}(\mathcal{Y})$ is a line segment connecting two adjacent extreme supported nondominated points if there is more than one nondominated point ($|\mathcal{Y}_N| > 1$). A maximally nondominated face can only have dimension zero if there is only one extreme supported nondominated point, implying that there is only one nondominated point. In other words, the Ideal point, where each objective attains its optimal value, is feasible. In the following, we assume that ($|\mathcal{Y}_N| > 1$). All $|\mathcal{Y}_{ES}|$ extreme points and precisely one corresponding extreme flow can be determined by using the enhanced parametric programming approach in $\mathcal{O}(\zeta + |\mathcal{Y}_{ES}| n(m + n \log n))$ time [Raith and Sedeño-Noda, 2017], where ζ denotes the time required to solve a given single-objective minimum cost flow problem. Also, the algorithm stores one extreme flow for each extreme nondominated point.

In Section 7.2, it is shown that for each maximal nondominated facet, we can create a reduced single-objective integer flow problem in which each image of a feasible flow lies on the maximal nondominated face. If we assume that we can solve the ACVF in output-polynomial time, we could determine for a maximal nondominated face all supported nondominated points whose images lie on this facet in output-polynomial time. Thus, we solve the ACVF on the reduced network with only the first objective function. Solving the ACVF successively for the reduced single-objective minimum cost integer flow problems for each $|\mathcal{Y}_{ES}| - 1$ maximal nondominated facets would yield all supported nondominated points for the given BOIMCF. Note that some points may lie in more than one maximal nondominated facet, namely the extreme supported nondominated points, and therefore, we have to ensure that we only store these flows once. In total, we would obtain all supported nondominated points in output-polynomial time. □

Theorem 7.12. *If we can determine all supported nondominated points for a bi-objective minimum cost integer flow problem in output-polynomial time, then we can solve the ACVF in output-polynomial time.*

Proof. Consider an instance of the ACVF. Construct a BOIMCF by setting the objective as $(c(f), -c(f))$ on the same instance. That is, for each arc $a \in A$, assign the cost vector $(c_a, -c_a)$. Any distinct flow vector remains distinct, is nondominated, and lies on the hyperplane $\{y : y_1 + y_2 = 0\}$. If we could determine all supported nondominated points in output-polynomial time, we could also solve the ACVF for the original single-objective MCIF. □

Let ACFV^{FIN} denote the decision problem of ACFV, i.e., given a set of distinct cost values $\{c(f^1), \ldots, c(f^k)\}$ of some feasible flows, decide if there exists a feasible flow f^p with a distinct cost value to the given ones $c(f^p) \notin \{c(f^1), \ldots, c(f^k)\}$. Suppose we can show that ACFV^{FIN} is not solvable in polynomial time. In that case, there does not exist an

output-polynomial time algorithm to determine all supported nondominated points unless $\mathbf{P} = \mathbf{NP}$.

Using these results, we can show that if we have given two distinct supported nondominated points y^1, y^2 with $y^1_1 < y^2_1$ for a bi-objective integer minimum cost flow problem (BOIMCF), it is **NP**-hard to decide if another supported nondominated point y^3 exists, which has higher cost in the first objective for y^1 and lower cost in the first objective for y^2.

Theorem 7.13. *Given two different supported nondominated points y^1 and y^2 of a BOIMCF with $y^1_1 < y^2_1$, the problem of deciding whether another supported nondominated point y^3 with $y^1_1 < y^3_1 < y^2_1$ exists, i.e., in between the supported nondominated point y^1 and y^2 is* **NP***-complete.*

Proof. The **NP**-completeness of the problem is established by reduction from the subset sum problem (SSP). Consider an instance of the (SSP) and construct the single-objective minimum cost integer flow problem (MCIF) as in the proof of Theorem 7.8, with the corresponding construction shown in Figure 7.6. An optimal flow for this instance has a cost of 0, while the highest cost of a feasible flow is $\sum_{i \in N} w_i$. Transforming the single-objective MCIF into a BOIMCF, as done in the proof of Theorem 7.12, ensures that all nondominated points are supported nondominated points and lie on the same edge between the two extreme supported nondominated points $(0,0)^\top$ and $\left(\sum_{i \in N} w_i, -\sum_{i \in N} w_i\right)^\top$. Introduce two arcs (s,t), one with cost $(k-1, -(k-1))$ and the other with $(k+1, -(k+1))$. This construction results in the two supported nondominated points $y_1 = (k-1, -(k-1))$ and $y_2 = (k+1, -(k+1))$. Any supported nondominated point y_3 satisfying $y^1_1 < y^1_3 < y^1_2$ would have a cost equal to k. Determining the existence of such a y_3 in polynomial time would allow solving the (SSP) in polynomial time, leading to a contradiction. □

Theorem 7.13 also implies that it is **NP**-hard to determine the next best supported nondominated point regarding the first or second objective for a given supported nondominated point of a BOIMCF. This means it is impossible to determine all supported nondominated points in a lexicographically ordered way regarding the first (or second) objective in output-polynomial time. However, this theorem is insufficient to show that the supported nondominated point could not be obtained in output-polynomial time, and it remains an open question.

Consider the BOIMCF of a subset sum instance in the proof of Theorem 7.13. We have shown that given a supported nondominated point, it is **NP**-hard to determine the next best supported nondominated point for this instance. However, for this instance of a BOIMCF, we can determine all supported-nondominated points in output-polynomial time using a similar algorithm to the pseudo-polynomial labeling algorithm presented in Pisinger [1999] to solve the subset sum problem .

7.4 Conclusion

This chapter discusses the time complexity of enumerating all supported nondominated points for MOIMCF. It concludes that there is no output-polynomial algorithm for a

MOIMCF problem with a fixed number of p objectives that determines all weakly supported nondominated points unless $\mathbf{P} = \mathbf{NP}$. Moreover, it shows that there cannot exist an output-polynomial time algorithm for the enumeration of all supported nondominated points that determine the points in an ordered manner in the outcome space unless $\mathbf{P} = \mathbf{NP}$. However, the question of whether an output-polynomial time algorithm exists remains open and could be tackled in future.

In contrast, this chapter presents output-polynomial time algorithms for determining all supported efficient solutions for BOIMCF problems and general MOIMCF problems with a fixed number of objectives. First, the approach determines all extreme supported nondominated points and the weighting vectors for each maximally nondominated face. Then, it successively determines all supported efficient solutions in the preimage of each maximally nondominated face by determining all optimal solutions for the corresponding single-objective parametric network flow problem using the all optimum flow algorithm recently presented in the previous chapter.

However, it might be that many supported efficient flows may be mapped to the same vector in the objective space. Thus, often, a minimal complete set (all nondominated points and one (efficient) preimage for each of them) is considered as a solution to a multi-objective optimization problem Serafini [1986]. The following chapter will present outcome and decision space methods for determining all nondominated points more efficiently.

Even though an output-polynomial time algorithm to determine all nondominated points for MOIMCF problems does not exist, even for the bi-objective case Bökler et al. [2017], future research could focus on new approaches to compute also nonsupported nondominated points/efficient solutions in bi- or even multi-objective MCF problems. Nonsupported solutions may be good compromise solutions and should thus not be neglected completely. Note that the difficulty in computing nonsupported solutions is not a specific property of multi-objective integer network flow problems but arises in many integer and combinatorial optimization problems and is one reason for their computational complexity, in general Ehrgott [2000] and Figueira et al. [2017]. One way to overcome this computational burden—at least to a certain degree—could be to determine nonsupported solutions only in regions of the Pareto front that are not well represented by the set of supported nondominated points.

8 Determining the Supported Nondominated Points for Bi-Objective Integer Minimum Cost Flow Problems

This chapter introduces novel methods for identifying supported nondominated points in bi-objective integer minimum cost flow (BOIMCF) problems, accompanied by a numerical comparison between decision-space and objective-space methods. A novel, equivalent, and more compact formulation of the minimum cost flow ILP formulation used in the ε-constraint scalarization approach is presented, demonstrating enhanced efficiency in the numerical tests.

The results presented in this Chapter are the outcome of joint work with Michael Stiglmayr and is available as a technical report Könen and Stiglmayr [2023], which has been submitted for publication.

8.1 Introduction

In addition to Algorithm 12 in Section 7.2, this chapter shows that the next best distinct cost flow can be determined in $\mathcal{O}(n^3)$ given an initial optimal flow. This result leads to an improved algorithm for enumerating all supported nondominated points for BOIMCF to Algorithm 12 if the number of branches needed is significantly smaller than the number of supported efficient solutions. An example is given in which the adjusted algorithm saves an exponential amount of considered flows to obtain all supported nondominated points.

While Algorithm 12 is a *decision-space* method, the chapter also introduces *objective-space* methods based on the ε-constraint scalarization for determining all supported nondominated points. Additionally, it presents a more compact and equivalent formulation for the ILP used in the ε-constraint method, improving computational efficiency. The ε-constraint approach can also be extended to get all nondominated points, even in higher dimensions, e.g., using generic scalarization based methods as discussed in Chapter 3. The adjusted algorithm can be seen as a combination of a decision-space and objective-space method.

The remainder of the chapter is structured as follows.In Section 8.2, we derive an adjusted algorithm to determine all supported nondominated points. In addition, in Section 8.3 a method is presented to use objective-space methods like the ε-constraint scalarization to determine all supported nondominated points. Section 8.4 presents a more compact formulation for the ILP used in the ε-constraint scalarization. Numerical results by these algorithms on different instances are reported in Section 8.5.

8.2 Adjusted Algorithm

In the current version of Algorithm 12, we search for a proper cycle in the reduced network to obtain a second supported efficient solution. Note that in the reduced network, any cycle would have a cost equal to zero regarding the weighted sum scalarization with a weight vector λ corresponding to the current maximal nondominated face F_i. However, we will now consider the first objective. We then search for a minimal proper cycle in the reduced network, i.e., a proper cycle with minimum cost regarding the first objective under all proper cycles in D'_λ. A proper zero cost cycle regarding the first objective in D'_λ would yield a supported efficient solution whose images map to the same vector as the initial solution (in the bi-objective case). Since we want, in the best case, only one flow per vector, we search for a minimal proper cost cycle under all proper cycles that have costs greater than zero. We then would obtain a distinct cost second-best flow regarding the first objective. Using the complementary slackness condition, we prove that we could obtain such a cycle in $\mathcal{O}(n^3)$. Using this fact, we could adjust the previously presented Algorithm 12 in Section 7.2 to find the supported nondominated points more efficiently.

We want to find all supported nondominated points for each maximally face F_i (in this case, given by an edge between two consecutive extreme points). For that, we want to determine all supported nondominated points between the edge of the two extreme points y^i and y^{i+1}. In Figure 7.3, an example is shown.

For y^i, let f' be a corresponding optimal flow. Considering the edge F_i, we know that f' is not only an optimal flow for the weight vector λ, but as well f' is an optimal flow regarding the first objective c^1 in D'_λ. So for f', there cannot be a negative cycle regarding the cost of c_1 in $D'_{\lambda,f'}$ and the complementary slackness conditions hold.

In order to find the next nondominated point on the edge F_i, we determine the minimal proper cycle $C \coloneqq \operatorname{argmin}\{c^1(C)\colon c^1(C) > 0,\ C \in D'_{\lambda,f'}\}$. Using techniques from Section 6.5 like Lemma 6.15 this can be done in $\mathcal{O}(n^3)$. Note that this is feasible only for the first and last supported nondominated point on a maximally nondominated face since it requires an optimal solution w.r.t. the first or second objective function to start with. Thus, this procedure cannot be extended to iteratively generate supported nondominated points along the face in an ordered way.

Let $\overline{D} = (\overline{d}_{ji})$ be the distance table of $D_{\lambda,f}$ concerning $\overline{c}^1(f)$, i.e., $\overline{d}_{ji}$ is the length of the shortest path P_{ji} from j to i in D_f with length $\overline{c}^1(f, P_{ji})$. The distance table may be computed in time $\mathcal{O}(n^3)$ using the Floyd-Warshall algorithm or in time $\mathcal{O}(n^2 \log n + mn)$ by (essentially) repeated calls to Dijkstra's algorithm; the latter is more efficient on sparse graphs Hamacher [1995].

Property 6.14 shows the following for proper minimal (i, j)-cycles. For any anti-parallel arc in A_f, i.e., $(i, j) \in A_f$ with $(j, i) \notin A_f$, the cost of a proper minimal (i, j)-cycle $C \in D_f$ is given by $c(f, C) = \overline{c}_{ij}(f) + \overline{d}_{ji}$.

Using the idea of Property 6.14 presented by Hamacher [1995], the problem is that it only applies to arcs with no anti-parallel arc in A_f. For all other arcs, the cost of the

corresponding proper minimal (i,j)-cycle has to be computed by finding a shortest path in $D_f \setminus \{(j,i)\}$. In the following result, we show that we can use the complementary slackness optimality conditions of an optimal solution to overcome this problem and restrict ourselves to consider only arcs with no anti-parallel arcs in $D_{\lambda,f}$.

Theorem 8.1. *We can determine the minimal cost cycle*

$$C \coloneqq \operatorname{argmin}_{C \in D'_{\lambda,f'}} \{c^1(C) \colon c^1(C) > 0\}$$

over all cycles with strictly positive weight in $\mathcal{O}(n^3)$.

Proof. Let C be a minimal proper cycle with the property $c^1(C) \neq 0$. So $\overline{c}^1(C,f) > 0$, because there cannot be a cycle with negative costs due to the negative cycle optimality conditions. Therefore, there exists an arc $(i,j) \in C$ with $\overline{c}_{ij}(f) > 0$. Since f is an optimal solution, the complementary slackness optimality conditions ensure that $\overline{c}^1_{ij}(f) > 0$ only holds if $(i,j) \in A$ and $f_{ij} = l_{ij}$ or $(j,i) \in D$ and $f_{ji} = u_{ji}$. Therefore, we have that $(i,j) \in \overline{A} \coloneqq \{(i,j) \in A_f \colon (i,j) \in A \text{ with } f_{ij} = l_{ij} \text{ or } (j,i) \in A \text{ with } f_{ji} = u_{ji}\}$ and $(j,i) \notin D_f$, (since the flow value of the corresponding arc of (i,j) in D is equal to the upper or lower capacity of this arc).

Since $(j,i) \notin D_f$ it holds that $\overline{c}^1(C,f) = \overline{c}_{ij} + \overline{d}_{ji}$ due to Property 6.14. Since $\overline{c}^1_{ij}(f) \geq 0$ any cycle C with $c^1(C)$ has at least one of such an arc. Consequently, we can determine a minimal proper cycle $C \coloneqq \operatorname{argmin}\{c^1(C) \colon c^1(C) > 0,\ C \in D'_{\lambda,f'}\}$ by just choosing $C = \operatorname{argmin}\{c(f,C) = \overline{c}_{ij} + \overline{d}_{ji} \colon C = \{(i,j)\} \cup P_{ji} \text{ with } (i,j) \in \overline{A}\}$. We can compute the distance table in $\mathcal{O}(n^3)$ time. Notice that this approach also provides the paths P_{ji} in addition to the cost $\overline{d}_{ji}$.

Given all pairwise distances $\overline{d}_{ji}$ and the reduced costs $\overline{c}^1_{ij}(f)$ for all (i,j) in D^1_f we can determine $c^1(f,C) = \overline{c}^1_{ij} + \overline{d}_{ji}$ in constant time $\mathcal{O}(1)$ and can determine the argmin in $\mathcal{O}(m)$ time. As a result, we can compute a minimal proper cycle with $c^1(C) \neq 0$ in time $\mathcal{O}(n^3)$. □

So let C be chosen as above. Let $f'' = f' + \chi(C)$. The flow f'' is supported, and its image lies on the edge F_i. We are going to prove that f'' is the next nondominated point on F_i regarding the cost of c^1, i.e., there does not exist a flow $\bar{f}$ whose image lies on F_i with the property $c^1(f') < c^1(\bar{f}) < c^1(f'')$.

Theorem 8.2. *There exists no flow $\bar{f}$ whose image $c^1(\bar{f})$ lies on F_1 between the images of f' and f'', i.e., $c^1(f') < c^1(\bar{f}) < c^1(f'')$.*

Proof. It holds that

$$f'' = f' + \chi(C) \qquad \text{and thus} \quad c^1(f'') = c^1(f') + c^1(C)$$

and we know that any flow $\bar{f}$ on F_1 can be written as

$$\bar{f} = f' + \sum_{C_i \in D'_{\lambda,f'}} \chi(C_i) \qquad c^1(\bar{f}) = c^1(f') + \sum_{C_i \in D'_{\lambda,f'}} c^1(C_i).$$

$$\implies c^1(f') < c^1(f') + c^1(C) = c^1(f'') \leq c^1(f') + \sum_{C_i \in D'_{\lambda,f'}} c^1(C_i) = c(\bar{f}).$$

The inequality $\sum_{C_i \in D'_{\lambda,f'}} c^1(C_i) \geq c^1(C)$ holds since the sum contains the positive cost of at least one cycle regarding the first objective. Since C was chosen as the minimal cost cycle

$$C \coloneqq \operatorname{argmin}_{C \in D'_{\lambda,f'}} \{c^1(C) \colon c^1(C) > 0\},$$

it holds that $c^1(f'') \leq c^1(\bar{f})$. □

Since we can compute the next supported nondominated point in D'_λ, we can now apply the binary partition approach as used in Hamacher [1995] and Section 6.5 using an arc with positve costs. Thereby, we substitute FindAnotherOptimalFlow in Algorithm 9 by Algorithm 15. This adjusted algorithm can be more efficient in determining all sup-

Algorithm 15: FindSecondDistinctCostBestFlow

Data: optimal flow f in P_{λ^i}
Result: A second distinct cost-best flow f' with $c(f') \neq c'(f)$, if one exists.

```
1 y ← ComputeNodePotential(f, D)
2 c̄ ← ComputeReducedCost(y, D)
3 (D̄, P) ← DetermineDistanceTableAndPaths(c̄, f, D)
4 C ← argmin{c(f, C)} = argmin_{C∈D'_{λ,f'}} {c^1(C) : c^1(C) > 0}
5 if C = null then return ∅
6 f' ← f + χ(C)
7 return f'
```

ported nondominated points than by determining all supported efficient flows, as the next example will show, since we might have an exponential number of supported efficient flows that we would not consider with this new technique.

Example 8.3. *Consider a graph D with nodes $\{1, \dots, k\}$, parameters $L, M \in \mathbb{N}$ and an optimal flow f w.r.t. the first objective. Node 1 has $b_1 = 2$ and node k has $b_k = -2$. For all other nodes $i \in \{2, \dots, k-1\}$ we have $b_i = 0$. The graph D contains arcs*

- *$(i, i+1)$ for all $i \in \{1, \dots, k-1\}$ with $u_{i,i+1} = (k-3)\,M + (L+2)$, $f_{i,i+1} = 2$ and reduced costs $\bar{c}^1_{i,i+1} = \bar{c}^2_{i,i+1} = 0$,*
- *arcs $(k-i, k-i-2)$ for all $i \in \{1, \dots k-3\}$ with $u_{k-i,k-i-2} = M$, $f_{k-i,k-i-2} = 0$ and $\bar{c}^1_{k-i,k-i-2} = \bar{c}^2_{k-i,k-i-2} = 0$, and*
- *one arc $(k, k-2)$ with $u_{k,k-2} = L$, $f_{k,k-2} = 0$ and $\bar{c}^1_{k,k-2} = 1, \bar{c}^2_{k,k-2} = -1$.*

Figure 8.1 illustrates this example for $k = 5$. There are $L+1$ nondominated points, which are all supported. However, we would have $(M+1)^{k-3}(L+1)$ efficient flows, which are all supported. However, using the branching technique described above, we would have only $L+1$ leaves at the end of our branching.

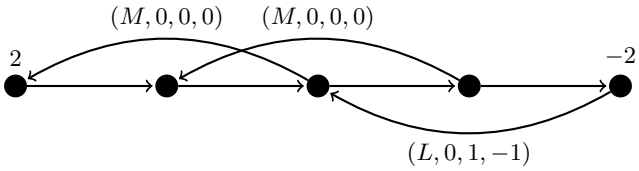


Figure 8.1: Graph D of the BMCIF in Example 8.3 with an optimal flow f w.r.t. the first objective. The arcs are labeled with $(u_a, f_a, \bar{c}^1_{ij}, \bar{c}^2_{ij})$. Here $l_{ij} = 0$ for all arcs and $(u_a = (k-3)M + (L+2), f_a = 2, \bar{c}^1_{ij} = 0, \bar{c}^2_{ij} = 0)$ for all arcs that are not labeled. The nodes are labeled with b_i and $b_i = 0$ for all nodes which are not labeled. $M > L$.

However, as shown in the following example, we might find supported efficient flows (even exponentially many) with images corresponding to supported nondominated points already found in other branches.

Example 8.4. *Consider a graph D with nodes $\{1, \dots, k\}$, parameter $L \in \mathbb{N}$ and an optimal flow f w.r.t. the first objective. Node 1 has $b_1 = 2$ and node k has $b_k = -2$. All other nodes $i \in \{2, \dots, k-1\}$ are transshipment nodes, $b_i = 0$. The graph D contains arcs*

- *$(i, i+1)$ for all $i \in \{1, \dots, k-1\}$ with $u_{i,i+1} = L + 2$, $f_{i,i+1} = 2$ and reduced costs $\bar{c}^1_{i,i+1} = \bar{c}^2_{i,i+1} = 0$, and*
- *$(k, k-i)$ for all $i \in \{2, \dots, k-1\}$ with $u_{k,k-1} = L$, $f_{k,k-1} = 0$ and reduced costs $\bar{c}^1_{k,k-1} = 1, \bar{c}^2_{k,k-i} = -1$.*

Figure 8.2 illustrates the graph D for $k = 5$. Then, this BMCIF has $L + 1$ nondominated points, and all of them are supported. However, we have $\sum_{i=0}^{L} \binom{(k-2)+i-1}{i}$ supported efficient flows, and all of them would yield a leave of our branching technique. This number can be exponential. Assume that $L > k$, then it holds that

$$\sum_{i=0}^{L} \binom{(k-2)+i-1}{i} > \sum_{i=0}^{k} \binom{k-3+i}{i} > \sum_{i=0}^{k} \binom{k-3}{i} = 2^{k-3}.$$

Since $k \in \mathcal{O}(m)$, we have an exponential amount of flows to consider.

8.3 Epsilon-Scalarizations on Reduced Networks

One decision-space method for determining nondominated points for general multi-objective linear programs is the well-known ε-constraint method [Haimes et al., 1971]. In the ε-constraint method, there is no aggregation of criteria as in the weighted sum scalarization. Instead, only one of the original objectives is minimized while the others are transformed into constraints.

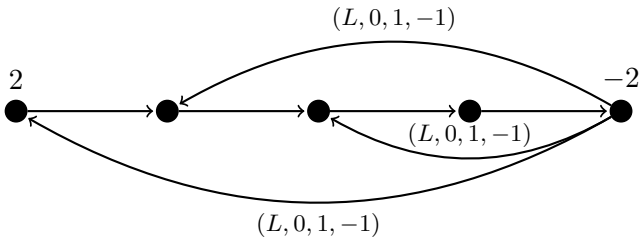


Figure 8.2: Graph D of the BMCIF in Example 8.4 with an optimal flow f w.r.t. the first objective. The arcs labeled with $(u_a, f_a, \bar{c}^1_{ij}, \bar{c}^2_{ij})$. Here $l_{ij} = 0$ for all arcs and $(u_a = L + 2, f_a = 2, \bar{c}^1_{ij} = 0, \bar{c}^2_{ij} = 0)$ for all arcs that are not labeled. The nodes are labeled with b_i and $b_i = 0$ for all nodes which are not labeled.

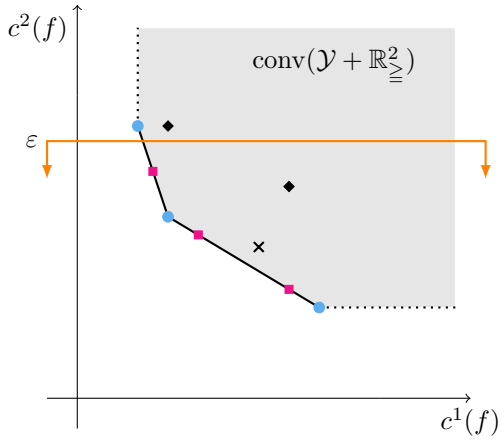


Figure 8.3: An example of the ϵ-constraint scalarization $\{\min_{f \in \mathcal{X}} c^1(f) \colon c^2(f) \leq \varepsilon\}$ for a BMCIF.

The ε-constraint scalarization of (MOIMCF) can be represented as:

$$\begin{aligned} \min \quad & c^j(f) \\ \text{s.t.} \quad & c^k(f) \leq \varepsilon_k \qquad \forall k \in \{1, \ldots, d\},\ k \neq j \\ & f \in \mathcal{X} \end{aligned}$$

An illustration of the ε-constraint for a bi-objective minimum cost integer flow problem can be found in Figure 8.3.

For the BOIMCF, all nondominated points can be found by solving a sequence of ε-constraint problems. Starting with the lexicographically minimal solution $\operatorname{lexmin}\{c^1(f), c^2(f)\}$ regarding the first objective, which can be determined in polynomial time, we can determine the next nondominated point using the ε-constraint. After each newly generated point, we update ε until the last nondominated point $\operatorname{lexmin}\{c^2(f), c^1(f)\}$ is generated. In Eusébio and Figueira [2009a], an implicit enumeration algorithm for BOIMCF is given, which solves such a sequence of ε-constraint problems by computing optimal non-integer solutions with a network simplex algorithm and then determining optimal integer solutions with a branch-and-bound technique.

We also could use this ε-constraint method to obtain the complete set of supported nondominated points for the BOIMCF, in which we have to solve one ε-constraint problem for each supported nondominated point. Thereby, we solve the ε-constraint method on each maximally nondominated face. Let $X_{D'}$ be the set of all feasible flows for the reduced network for one maximal nondominated face, i.e., all solutions whose image lies on this maximal nondominated face. Any solution in the reduced network is a supported efficient solution. Using the ε-constraint method on all of these facets would determine all supported nondominated points. We only must solve one ε-constraint problem for each supported nondominated vector. Let $T(\varepsilon)$ be the longest time required to solve such a ε-constraint problem.

Theorem 8.5. *For a given BOIMCF, we can determine all S supported nondominated points in time $\mathcal{O}(|\mathcal{Y}_{ES}|\, n(m + nlogn) + \zeta + S(T(\varepsilon))$.*

Proof. The enhanced parametric network approach [Raith and Ehrgott, 2009] requires $\mathcal{O}(|\mathcal{Y}_{ES}|n(m + n\log n) + \zeta)$ time, where ζ is the time required to solve a single-objective minimum cost integer flow problem, and $|\mathcal{Y}_{ES}|$ is the number of extreme supported nondominated points. Since the algorithm determines the extreme points in decreasing order of $c_1(f)$, no additional time is needed for sorting. Defining the weight vectors λ^i for $i = 1, \ldots, |\mathcal{Y}_{ES}| - 1$ and constructing the network with the corresponding cost function takes $\mathcal{O}(|\mathcal{Y}_{ES}|(n + m))$ time. Determining all S_i supported nondominated points on the maximally nondominated face F_i requires solving $S_i - 1$ ε-constraint problems. Doing so for each maximally nondominated face requires $\mathcal{O}(ST(\varepsilon))$ time. □

8.4 A more Compact Formulation for the ILP in the Epsilon-Constraint Method

The ε-constraint problem contains m variables and $2\,m + n + 1$ constraints. However, according to Theorem 6.21, each flow f^* can be written by an initial optimal tree solution f and a conical combination of incidence vectors of all *induced cycles* with bounded coefficients, i.e.,

$$f^* = f + \sum_{a \notin T} \lambda_a \, \chi(C_a)$$

for some $\lambda \in \mathbb{Z}$ and it holds that

$$c(f^*) = c(f) + \sum_{a \notin T} \lambda_a \, c(C_a).$$

Therefore, instead of solving a constrained minimum cost integer flow problem, we could also solve the following ILP, which searches for the best combination of induced cycles

such that the capacity constraints are satisfied.

$$\begin{aligned} \min \quad & \sum_{a \notin T} \lambda_a \, c^1(C_a) && \text{(c-MOIMCF)} \\ \text{s.t.} \quad & 0 - f_{ij} \leq \sum_{a:(i,j) \in C_a} \lambda_a \, \chi_{ij}(C_a) \leq u_{ij} - f_{ij} \quad \forall (i,j) \in \bigcup_{a \notin T} C_a \\ & \sum_{a \notin T} \lambda_a \, c^2(C_a) \leq \varepsilon \end{aligned}$$

Theorem 8.6. *For ε sufficiently large, the set of feasible solutions of* (MOIMCF) *and* (c-MOIMCF) *coincide.*

Proof. Let f be an initial optimal tree solution. Assume that f^* is a feasible solution for MOIMCF. After Theorem 6.21, the solution f^* can be written as $f^* = f + \sum_{a \notin T} \lambda_a^* \, \chi(C_a)$ for some $\lambda \in \mathbb{Z}$. Let λ_a^* for $a \notin T$ be the solution for c-MOIMCF, then $0 - f_{ij} \leq \sum_{a:(i,j) \in C_a} \lambda_a \, \chi_{ij}(C_a) \leq u_{ij} - f_{ij}$ for all $(i,j) \in \bigcup_{a \notin T} C_a$ would be satisfied, since otherwise an arc (u,v) exists where the capacity constraint $0 \leq f_{uv}^* \leq u_{uv}$ would not hold. A contradiction of the feasibility of f^*. Now assume that λ_a' for $a \notin T$ is a feasible solution of c-MOIMCF. Let $f' = f + \sum_{a \notin T} \lambda_a' \, \chi(C_a)$. Since f was an initial optimal tree solution and we only change flow on cycles, it holds that $\sum_{j:(i,j) \in A} f_{ij}' - \sum_{j:(j,i) \in A} f_{ji}' = b_i$ for all $i \in V$. The boundaries of $0 \leq f_{ij} \leq u_{ij}$ for all $(i,j) \in A$ are also satisfied, since otherwise $u_{ij} - f_{ij}$ for one arc $(i,j) \in \bigcup_{a \notin T} C_a$ would not be satisfied. A contradiction.

□

This ILP has n variables and at least $n-1$ constraints less than the standard ε-constraint problem. According to Theorem 6.21, the respective sets of feasible solutions coincide. In Section 8.5, we evaluate numerically which approach is faster in practice, the standard ε-constraint method or the combination approach of the induced cycles. We also compare the running times to determine all supported nondominated points compared to Algorithm 12 in Section 7.2 and Section 8.2.

8.5 Numerical Experiments

This section presents the implementation and numerical evaluation of the four methods. The section aims to report and compare the results, providing a comprehensive understanding of their behaviors.

All computations are conducted on a computer with an Intel® Core™i8-8700U CPU 3.20 GHz processor with 32 GB RAM, using a LINUX operating system. The algorithms are implemented and run in Python (Version 3.11). In addition, for solving the ε-constraint scalarizations, Gurobi 12.1 embedded in Python is used. To ensure fair comparisons, the ILPs in Gurobi were solved using a single thread.

For the computational experiments, we utilized test instances from Figure 8.1 and Figure 8.2, as well as minimum cost integer flow problem classes generated by the NETGEN network generator [Klingman et al., 1974]. The entire test comprised 11 problem classes,

with each class consisting of a set of 15 randomly generated network problems. The parameters that allowed the random generation of each NETGEN instance are the number of nodes, arcs, and nodes acting as supply or sink nodes, respectively, the greatest cost, greatest capacity, and the total supply in each network. In each problem class, the number of arcs and nodes varies. The instances ranged from 50 to 5000 nodes with 100 to 10000 arcs. These variables were chosen as independent variables due to their direct influence on the number of possible supported nondominated points or supported efficient solutions and, therefore, influencing the different number of iterations of the different methods. All the other parameters that NETGEN can accept were kept constant. All instances have two nodes acting as supply nodes and two as sink nodes, a maximum arc cost of 10 for both objective functions, a maximum upper capacity of 50, and a total supply of 50.

Results for the NETGEN instance classes are presented in Table 8.1, while the results of the test instances from Figure 8.1 and Figure 8.2 are summarized in Table 8.2 and Table 8.3. These tables display the number of extreme supported nondominated points, supported nondominated points, supported efficient solutions, and CPU time for all four methods. We display the min, max, and mean of the times and numbers of the 15 network problems. Note that we only consider problem instances with at least two nondominated points.

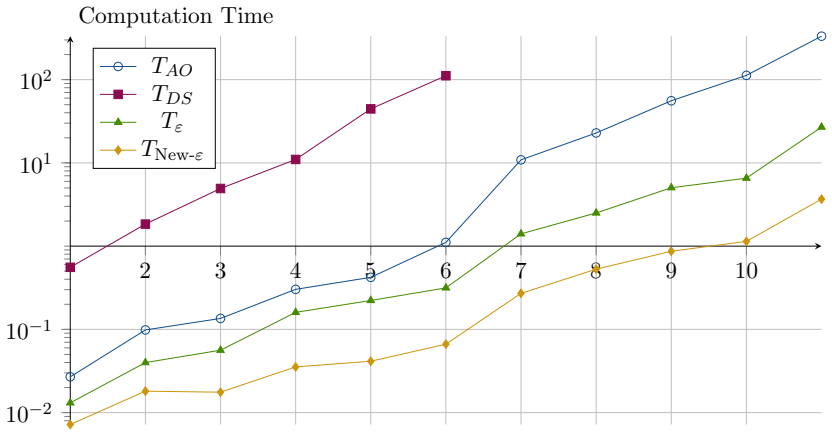


Figure 8.4: Line graph showing the comparison of average computation times for the different methods across the problem instances from Table 8.1. The x-axis represents different problem instances, while the y-axis (logarithmic scale) indicates the computation time.

Table 8.2 indicates, as expected, that the adjusted algorithm outperforms the all optimal flow algorithm when the number of branches needed is significantly smaller than the number of supported efficient solutions. However, when the number of branches needed equals the number of efficient solutions, the all optimal algorithm outperforms the adjusted algorithm, as evident in Table 8.3. The high computational cost of the Floyd-Warshall algorithm used in each branch contributes to the adjusted algorithm's suboptimal performance, as highlighted in Table 8.1, and indicates that this algorithm performs poorly in practice. Despite this, when the number of supported nondominated points and sup-

ported efficient solutions equals, and therefore, the all optimal flow algorithm would run in output-polynomial time to determine the number of all supported nondominated points, the objective-space methods clearly outperform this algorithm.

The observed differences in running times of the outcome space methods against the decision space methods can, in part, be attributed to the nature of the implementations. Gurobi, a highly optimized solver, benefits from extensive engineering and decades of refinement. In contrast, the other algorithms in our study were implemented from scratch in Python, which inherently results in slower execution times due to the interpreted nature of the language and the absence of low-level optimizations.

It is important to emphasize that developing a high-performance, C-based implementation of our proposed algorithms is beyond the scope of this chapter. However, our results still provide valuable insights into the relative efficiency of different approaches independent of implementation-specific optimizations. It is important to note that despite the existence of high-performance implementations for outcome space methods, the development of new decision space methods should not be overlooked.

To foster reproducibility and further research, our code and benchmark instances can be found in an open repository under the following link to allow other researchers to replicate our findings, test alternative implementations, and explore further optimizations.

On a positive note, the more compact formulation of the ε-constraint scalarization surpasses the classic formulation in each instance, as shown in Table 8.4. The mean CPU time needed in the more compact formulation for the ε-constraint scalarization across all instance classes is 4.244 times faster than the standard formulation, demonstrating its efficiency. The time gap may increase in instances including more nodes and arcs. This is a nice result since the more compact formulation can also be used in any ε-constraint scalarization method for multi-objective minimum cost integer flow problems with $d \geq 3$ objectives or the determination of all nondominated points for minimum cost integer flow problems.

In conclusion, the computational experiments provide valuable insights into the strengths and weaknesses of the implemented algorithms. The more compact formulation of ε-constraint scalarization presents a promising direction for future research, offering faster computation times and potential applications in multi-objective minimum cost integer flow problems with three or more objectives.

8.6 Conclusion

This chapter proposes novel methods for identifying supported nondominated points in bi-objective minimum cost flow problems accompanied by a numerical comparison between decision- and objective-space methods. A novel, equivalent, and more compact formulation of the minimum cost flow ILP formulation used in the ε-constraint scalarization approach is introduced.

The numerical tests show that the outcome space methods clearly outperform the decision-space methods, even if they do not run in output-polynomial time. The compact

formulation of the ILP for the ε-method shows a significant time improvement compared to the conventional ILP. The new formulation can also be used to compute all nondominated points in multidimensional minimum cost integer flow problems and could be investigated in the future.

For MOIMCF, supported nondominated points are often only a minor part of the complete set of nondominated points, even for the bi-objective case. However, determining supported nondominated points is often needed as a first step in two-phase exact methods and for population-based heuristics. The development of improved two-phase methods that compute all nondominated points for MOIMCF could be explored in the future.

Table 8.1: Numerical results for the different instance classes generated with NETGEN. T-$|\mathcal{Y}_{ES}|$ displays the time needed to determine the number of extreme points. T-AO, T-DS, T-ε, T-New-ε refers to the CPU time in seconds needed for Algorithm 12, the adjusted version with Algorithm 15, the ε-Method, and the new more compact ε-Method. Empty entries (–) reflect a CPU time of over 500 seconds and have not been recorded.

Class		$\lvert\mathcal{Y}_{ES}\rvert$	T-$\lvert\mathcal{Y}_{ES}\rvert$	$\lvert\mathcal{Y}_S\rvert$	$\lvert\mathcal{X}_S\rvert$	T-AO	T-DS	T-ε	T-New-ε
1	min	2	0.027	4	4	0.0058	0.1010	0.0049	0.0027
$n = 50$	max	10	0.163	41	47	0.0887	1.1871	0.0287	0.0161
$m = 100$	mean	4.9	0.074	20.7	22.2	0.0269	0.5565	0.0131	0.0072
2	min	4	0.125	23	29	0.0408	0.7778	0.0156	0.0097
$n = 50$	max	15	0.589	97	227	0.3906	8.0199	0.1056	0.0296
$m = 200$	mean	9.2	0.340	42.3	58.5	0.0984	1.8382	0.0399	0.0181
3	min	2	0.103	3	3	0.0123	0.3989	0.0045	0.0027
$n = 100$	max	14	0.912	49	99	0.3557	15.524	0.1011	0.0365
$m = 200$	mean	6.47	0.389	25.3	36.1	0.1353	4.9246	0.0562	0.0176
4	min	6	0.718	20	20	0.1110	4.2690	0.0634	0.0164
$n = 100$	max	36	4.959	128	154	0.8094	33.2009	0.3805	0.1197
$m = 400$	mean	12.6	1.488	53.6	68.6	0.3025	10.9926	0.1601	0.0354
5	min	4	0.825	4	4	0.0701	4.9458	0.0823	0.0140
$n = 200$	max	17	3.876	86	86	1.1996	109.7116	0.4540	0.0794
$m = 400$	mean	9.4	2.164	35.8	36.6	0.4217	44.5244	0.2223	0.0414
6	min	2	0.669	5	5	0.0625	5.4639	0.0311	0.0052
$n = 200$	max	24	10.279	128	311	4.4925	380.9636	0.6182	0.1250
$m = 800$	mean	13.4	5.379	64.6	82.4	1.1090	111.4111	0.3150	0.0664
7	min	4	19.586	10	10	3.2243	–	0.4168	0.0870
$n = 1000$	max	18	102.092	64	64	18.9144	–	2.3113	0.4913
$m = 2000$	mean	11.8	62.721	32.6	33.4	10.8732	–	1.4042	0.2709
8	min	4	39.428	7	7	2.6272	–	0.4336	0.0963
$n = 1000$	max	32	341.543	112	121	41.6134	–	4.0621	0.9461
$m = 4000$	mean	18.6	195.961	55.4	61.4	22.8661	–	2.5020	0.5284
9	min	7	75.994	9	9	16.1292	–	1.3056	0.2225
$n = 2000$	max	24	516.131	73	124	139.9464	–	9.3591	1.5732
$m = 4000$	mean	14.8	300.958	38.6	48.6	55.7729	–	5.0278	0.8677
10	min	9	359.654	17	17	27.3013	–	0.5476	0.5476
$n = 2000$	max	26	1090.674	142	220	279.7599	–	10.7284	1.9579
$m = 8000$	mean	17.2	716.485	67.6	80.8	112.5519	–	6.5519	1.1404
11	min	6	777.874	14	14	106.6934	–	11.3873	1.4136
$n = 5000$	max	21	2741.023	110	160	1010.9983	–	44.9173	6.2393
$m = 10000$	mean	13.6	1666.410	47.9	48.34	332.6101	–	26.8930	3.6720

Table 8.2: Numerical results for the different instances of Figure 8.1. Each instance has $|\mathcal{Y}_{ES}| = 2$ extreme supported nondominated and $|\mathcal{Y}_S| = 6$ supported nondominated points. For the first 5 instances, it holds $M = 10$, $L = 5$. For the last, it holds $M = 1, L = 5$. In these instances the adjusted algorithm needs only $|\mathcal{Y}_S| = 6$ branches.

Class	$\lvert\mathcal{X}_S\rvert$	T-AO	T-DS	T-ε	T-New-ε
$n = 5$	726	0.0587	0.0012	0.0018	0.0013
$n = 6$	7986	0.6490	0.0016	0.0017	0.0012
$n = 7$	87846	7.8305	0.0019	0.0223	0.0013
$n = 8$	966306	95.4091	0.0025	0.0193	0.0013
$n = 9$	10629366	1133.3333	0.0028	0.0191	0.0013
$n = 20$	786432	165.3812	0.0136	0.0219	0.0020

Table 8.3: Numerical results for the different instances of Figure 8.2. Each instance has $|\mathcal{Y}_{ES}| = 2$ extreme supported nondominated and $|\mathcal{Y}_S| = 6$ supported nondominated points. For the first six instances, it holds $L = 5$. For the last $L = 3$. In this instances the adjusted algorithm needs exactly $|\mathcal{X}_S|$ branches.

Class	$\lvert\mathcal{X}_{SN}\rvert$	T-AO	T-DS	T-ε	T-New-ε
$n = 5$	56	0.0048	0.0120	0.0016	0.0012
$n = 6$	126	0.0133	0.0356	0.0016	0.0012
$n = 7$	252	0.0427	0.0864	0.0019	0.0013
$n = 8$	462	0.0641	0.1954	0.0020	0.0014
$n = 9$	792	0.1257	0.4160	0.0021	0.0014
$n = 10$	1287	0.2221	0.8041	0.0202	0.0016
$n = 20$	33649	12.9121	84.201	0.0195	0.0027

Table 8.4: Displays the time difference of the ε-epsilon method versus the more compact formulation of the mean for each randomly generated NETGEN class instances of Table 8.1.

Class	T-ε / T-New-ε
1	1.82
2	2.20
3	3.19
4	4.52
5	5.37
6	4.74
7	5.18
8	4.74
9	5.79
10	5.73
11	7.87

9 Conclusion

This thesis has explored the complexities and challenges of supportedness and determining representations or the entire nondominated point set in multi-objective combinatorial optimization, focusing mainly on multi-objective integer minimum cost flow problems.

The thesis presents exact generic scalarization-based algorithms for MOCO problems. These methods decompose the problem into a series of scalarized single-objective subproblems, which can be solved using existing single-objective (IP-)solvers. Various implementation strategies are examined, including choices regarding the sequence of subproblems, scalarization techniques, and decisions about computing the entire nondominated point set or a representative subset.

However, MOCO and MOIMCF belong to the class of computationally intractable problems, often containing a huge set of nondominated points. This results in high computational effort, particularly for high-dimensional problems or large instances, highlighting the need for approximation techniques and alternative methods to represent the entire nondominated point set.

Therefore, this thesis focuses on determining subsets of the entire nondominated point set and investigates the existence of output-polynomial time algorithms for different solution concepts in MOIMCF problems. A key finding of this thesis is the role of supported nondominated points as high-quality representations of the complete nondominated point set in MOIMCF. Across various test instances, supported solutions consistently demonstrated superior representational quality, as measured by hypervolume ratio and coverage error. For all instances, the hypervolume ratio of the supported nondominated points as representations always are close to one and provides minor coverage errors. In contrast, extreme supported nondominated points yielded significantly lower quality measures, particularly as arc capacities increased. On average, across 11 three-dimensional test classes comprising 15 instances each, the hypervolume ratio of supported nondominated points was 16% higher than that obtained from extreme supported nondominated points. Furthermore, the mean coverage error of supported points was only 28% of that of extreme supported points, further demonstrating their superior representational quality.

Beyond their representational advantages, the supported nondominated points are more straightforward to compute than the nonsupported ones and can serve as a foundation for two-phase methods. Despite the importance of supportedness, several different characterizations for supported efficient solutions (and supported nondominated points) are used in the literature. This thesis addresses these inconsistencies in the literature regarding definitions of supported nondominated points. Through theoretical analysis and counterexamples, it demonstrates that while various definitions are equivalent in multi-objective linear problems, they produce distinct sets of supported nondominated points in combinatorial settings, leading to structural and computational properties differences.

This thesis formally introduces the distinction between supported and weakly supported efficient solutions.

The thesis further examines the computational complexity of enumerating all supported nondominated points for MOIMCF. It concludes that there is no output-polynomial algorithm for a MOIMCF problem with a fixed number of p objectives that determines all weakly supported nondominated points unless $\mathbf{P} = \mathbf{NP}$. Moreover, it shows that there cannot exist an output-polynomial time algorithm for the enumeration of all supported nondominated vectors that determine the vectors in an ordered manner in the outcome space unless $\mathbf{P} = \mathbf{NP}$. However, the question of whether an output polynomial time algorithm exists remains open and could be tackled in the future.

In contrast, this thesis presents output-polynomial time algorithms for determining all supported efficient solutions for BOIMCF problems and general MOIMCF problems with a fixed number of objectives. It also proposes novel methods for identifying supported nondominated points in bi-objective minimum cost flow problems accompanied by a numerical comparison between decision- and objective-space methods. The numerical tests highlight that the outcome space methods clearly outperform the decision-space methods, even if they do not run in output-polynomial time. The compact formulation of the ILP for the ε-method shows a significant time improvement compared to the conventional ILP. The new ILP formulation for the ε-method demonstrates significant time improvements over the conventional ILP formulation and may also be useful for computing all nondominated points in MOIMCF, and should be investigated in future.

Looking forward, several promising research directions arise from this work. Enhancing parallel computing techniques could improve the efficiency of defining point algorithms and other generic scalarization-based methods for MOCO problems, addressing scalability concerns in high-dimensional settings. Additionally, structured combinatorial approaches for selectively computing nonsupported nondominated solutions in underrepresented regions of the Pareto front could mitigate computational challenges while preserving solution quality. Moreover, extending the characterization of supportedness to broader multi-objective optimization contexts remains an open challenge.

For MOIMCF specifically, future research could explore structured combinatorial approaches to efficiently analyze the conical combinations of induced cycles, potentially leading to improved algorithms for network flow problems. Given the significant time improvements observed with the new ILP formulation for the ε-constraint method, it would be valuable to investigate whether this formulation can enhance computational efficiency when integrated into generic scalarization-based methods, such as the defining point algorithm for MOIMCF problems.

Although supported nondominated points yield high-quality representations, they are still numerous (however, being only a minor part of the complete entire set of nondomiated points). Future research could explore whether a strategically chosen subset of the supported nondominated points, potentially derived from intermediate solutions, could provide sufficiently high-quality representations while reducing computational effort.

Even though an output-polynomial time algorithm to determine all nondominated points for MOIMCF problems does not exist, even for the bi-objective case, future research could

focus on new approaches to compute also nonsupported nondominated points/efficient solutions in bi- or even multi-objective MCF problems. Nonsupported solutions may be good compromise solutions and should thus not be overlooked entirely. Furthermore, the open question of whether supported nondominated points can be determined in output-polynomial time remains an important direction of research.

In summary, this thesis significantly contributes to the study of MOCO and especially MOIMCF by advancing algorithmic methodologies, refining theoretical foundations, and offering insights into computational complexity. By addressing gaps in the literature and proposing novel approaches, it lays the groundwork for further advancements in MOIMCF research.

Bibliography

Ahuja, R. K., T. L. Magnanti, and J. B. Orlin (Feb. 1993). *Network Flows: Theory, Algorithms, and Applications*. 1. Prentice Hall.

Alrabeeah, M., A. Al-Hasani, S. Kumar, and A. Eberhard (Mar. 2020). "Enhancement of the improved recursive method for multi-objective integer programming problem". In: *Journal of Physics: Conference Series* 1490, p. 012061. DOI: 10.1088/1742-6596/1490/1/012061.

Aneja, Y. and K. Nair (1979). "Bicriteria Transportation Problem". In: *Management Science* 25, pp. 73–78. DOI: 10.1287/mnsc.25.1.73.

Argyris, N., J. R. Figueira, and A. Morton (2011). "Identifying preferred solutions to multi-objective binary optimisation problems, with an application to the Multi-Objective Knapsack Problem". In: *Journal of Global Optimization* 49, pp. 213–235. DOI: 10.1007/s10898-010-9541-9.

Baste, J., M. R. Fellows, L. Jaffke, T. Masarík, M. O. de Oliveira, G. Philip, and F. A. Rosamond (2019). "Diversity in combinatorial optimization". In: *CoRR* abs/1903.07410. arXiv: 1903.07410.

Battista, G. D., P. Eades, R. Tamassia, and I. G. Tollis (1998). *Graph Drawing: Algorithms for the Visualization of Graphs*. 1st. USA: Prentice Hall PTR.

Bauß, J., S. N. Parragh, and M. Stiglmayr (2024). "On improvements of multi-objective branch and bound". In: *EURO Journal on Computational Optimization* 12, p. 100099. DOI: 10.1016/j.ejco.2024.100099.

Bazgan, C., F. Jamain, and D. Vanderpooten (2017). "Discrete representation of the non-dominated set for multi-objective optimization problems using kernels". In: *European Journal of Operational Research* 260.3, pp. 814–827. DOI: 10.1016/j.ejor.2016.12.038.

Bektas, T. (2018). "Disjunctive programming for multiobjective discrete optimisation". In: *INFORMS Journal on Computing*, pp. 1–18. DOI: 10.1287/ijoc.2017.0804.

Benson, H. P. (1978). "Existence of efficient solutions for vector maximization problems". In: *Journal of Optimization Theory and Applications* 26.4, 569–580. DOI: 10.1007/BF00933152.

Bertsekas, D. P. (1998). *Network Optimization. Continuous and Discrete Methods*. Athena Scientific, p. 593.

Boissonnat, J.-D., M. Sharir, B. Tagansky, and M. Yvinec (1998). "Voronoi diagrams in higher dimensions under certain polyhedral distance functions". In: *Discrete & Computational Geometry* 19, pp. 485–519. DOI: 10.1007/PL00009366.

Boland, N., H. Charkhgard, and M. Savelsbergh (2016). "The L-shape search method for triobjective integer programming". In: *Mathematical Programming Computation* 8, pp. 217 –251. DOI: 10.1007/s12532-015-0093-3.

Boland, N., H. Charkhgard, and M. Savelsbergh (2017a). "A new method for optimizing a linear function over the efficient set of a multiobjective integer program". In: *European Journal of Operational Research* 260.3, pp. 904 –919. DOI: 10.1016/j.ejor.2016.02.037.

Boland, N., H. Charkhgard, and M. Savelsbergh (2017b). "The Quadrant Shrinking Method: A simple and efficient algorithm for solving tri-objective integer programs". In: *European Journal of Operational Research* 260.3, pp. 873 –885. DOI: 10.1016/j.ejor.2016.03.035.

Boyd, S. and L. Vandenberghe (2004). *Convex Optimization*. Cambridge university press.

Bökler, F. (Apr. 2018). "Output-sensitive Complexity of Multiobjective Combinatorial Optimization Problems with an Application to the Multiobjective Shortest Path Problem". PhD thesis.

Bökler, F., M. Ehrgott, C. Morris, and P. Mutzel (Jan. 2017). "Output-sensitive complexity of multiobjective combinatorial optimization". In: *Journal of Multi-Criteria Decision Analysis* 24.1-2, 25–36. DOI: 10.1002/mcda.1603.

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. Berlin, Heidelberg: Springer Berlin Heidelberg, 288–299. DOI: 10.1007/978-3-662-48350-3_25.

Bökler, F., S. N. Parragh, M. Sinnl, and F. Tricoire (2024). "An outer approximation algorithm for generating the Edgeworth–Pareto hull of multi-objective mixed-integer linear programming problems". In: *Mathematical Methods of Operations Research* 100.1, 263–290. DOI: 10.1007/s00186-023-00847-8.

Calvete, H. I. and P. M. Mateo (1996). "A sequential network-based approach for the multiobjective network flow problem with preemptive priorities". In: *Multi-Objective Programming and Goal Programming - Theory and Applications*. Ed. by M. Tamiz. Vol. 432. Lecture Notes in Economics and Mathematical Systems. Berlin: Springer Verlag, pp. 74–86. DOI: 10.1007/978-3-642-87561-8_7.

Camerini, P. M., G. Galbiati, and F. Maffioli (1984). "The complexity of multi-constrained spanning tree problems". In: *Theory of Algorithms*. Ed. by L. Lovasz. North-Holland, Amsterdam, pp. 53–101.

Chalmet, L., L. Lemonidis, and D. Elzinga (1986). "An algorithm for the bi-criterion integer programming problem". In: *European Journal of Operational Research* 25, pp. 292–300. DOI: 10.1016/0377-2217(86)90093-7.

Chlumsky-Harttmann, F. (2025). "Robust Multi-Objective Optimization: Analysis and Algorithmic Approaches". doctoralthesis. Rheinland-Pfälzische Technische Universität Kaiserslautern-Landau, pp. XI, 109. DOI: 10.26204/KLUED0/8859.

Christofides, N. and V. Valls (1986). "Finding all optimal solutions to the network flow problem". In: *Netflow at Pisa*. Ed. by G. Gallo and C. Sandi. Berlin, Heidelberg: Springer Berlin Heidelberg, 209–212. DOI: 10.1007/BFb0121096.

Cook, S. A. (1971). "The complexity of theorem-proving procedures". In: *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, pp. 151–158. DOI: 10.1145/800157.805047.

Cook, W. J., W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver (1998). *Combinatorial Optimization*. John Wiley & Sons, New York.

Cormen, T. H., C. E. Leiserson, R. L. Rivest, and C. Stein (2001). *Introduction to Algorithms*. 2nd. The MIT Press.

Correia, P., L. Paquete, and J. R. Figueira (2021). "Finding multi-objective supported efficient spanning trees". In: *Computational Optimization and Applications* 78.2, 491–528. DOI: 10.1007/s10589-020-00251-6.

Dächert, K., J. Gorski, and K. Klamroth (2012). "An augmented weighted Tchebycheff method with adaptively chosen parameters for discrete bicriteria optimization problems". In: *Computers and Operations Research* 39, pp. 2929–2943. DOI: 10.1016/j.cor.2012.02.021.

Dächert, K. and K. Klamroth (2015). "A linear bound on the number of scalarizations needed to solve discrete tricriteria optimization problems". In: *Journal of Global Optimization* 61.4, pp. 643–676. DOI: 10.1007/s10898-014-0205-z.

Dächert, K., K. Klamroth, R. Lacour, and D. Vanderpooten (2017). "Efficient computation of the search region in multi-objective optimization". In: *European Journal of Operational Research* 260.3, pp. 841 –855. DOI: 10.1016/j.ejor.2016.05.029.

Dai, R. and H. Charkhgard (2018). "A two-stage approach for bi-objective integer linear programming". In: *Operations Research Letters* 46.1, pp. 81–87.

Dantzig, G. B. (1947). "Maximization of a linear function of variables subject to linear inequalities". In: *Quarterly of Applied Mathematics* 5, pp. 183–187.

Dantzig, G. B. (1951). "Application of the simplex method to a transportation problem". In: *Activity Analysis and Production and Allocation*. Ed. by T. C. Koopmans. Wiley, 359–373.

Dantzig, G. B. and L. R. Ford (1956). "A primal-dual algorithm for linear programs". In: *Linear Inequalities and Related Systems* 38, pp. 171–181. DOI: 10.1515/9781400881987-008.

Dhaenens, C., J. Lemesre, and E.-G. Talbi (2010). "K-PPM: A new exact method to solve multi-objective combinatorial optimization problems". In: *European Journal of Operational Research* 200, pp. 45–53. DOI: 10.1016/j.ejor.2008.12.021.

Diestel, R. (Mar. 2006). *Graphentheorie*. 3., neu bearb. und erw. A. Springer, Berlin.

Domínguez-Ríos, M. Ángel, F. Chicano, and E. Alba (2021). "Effective anytime algorithm for multiobjective combinatorial optimization problems". In: *Information Sciences* 565, 210–228. DOI: 10.1016/j.ins.2021.02.074.

Dächert, K., K. Klamroth, and D. Könen (forthcoming). "Generic scalarization based algorithms". In: *Handbook of Multi-objective Combinatorial Optimization*. Ed. by M. Ehrgott, J. R. Figueira, and L. Paquete. To appear. Springer.

Dächert, K. (2014). "Adaptive Parametric Scalarizations in Multicriteria Optimization". Dissertation. Bergische Universität Wuppertal.

Dächert, K., T. Fleuren, and K. Klamroth (2024). "A simple, efficient and versatile objective space algorithm for multiobjective integer programming". In: *Mathematical Methods of Operations Research*. DOI: 10.1007/s00186-023-00841-0.

Edwin Romeijn, H. and R. L. Smith (1999). "Parallel algorithms for solving aggregated shortest-path problems". In: *Computers & Operations Research* 26.10, 941–953. DOI: 10.1016/S0305-0548(99)00029-5.

Ehrgott, M. and A. Skriver (2003). "Solving biobjective combinatorial max-ordering problems by ranking methods and a two-phases approach". In: *European Journal of Operational Research* 147, pp. 657–664. DOI: 10.1016/S0377-2217(02)00353-3.

Ehrgott, M. (2000). "Hard to say it's easy – Four reasons why combinatorial multiobjective programmes are hard". In: *Research and Practice in Multiple Criteria Decision Making*. Ed. by Y. Y. Haimes and R. E. Steuer. Vol. 487. Lecture Notes in Economics and Mathematical Systems. Springer Berlin Heidelberg, pp. 69–80.

Ehrgott, M. (2005). *Multicriteria optimization*. Vol. 491. Springer Science & Business Media.

Ehrgott, M. (2006). "A discussion of scalarization techniques for multiple objective integer programming". In: *Annals of Operations Research* 147.1, 343–360. DOI: 10.1007/s10479-006-0074-z.

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, 757–778. DOI: 10.1007/s10898-011-9709-y.

Ehrgott, M. and S. Ruzika (2008). "Improved ε-Constraint Method for Multiobjective Programming". In: *Journal of Optimization Theory and Applications* 138, pp. 375–396. DOI: 10.1007/s10957-008-9394-2.

Ehrgott, M. and D. Tenfelde-Podehl (2003). "Computation of ideal and Nadir values and implications for their use in MCDM methods". In: *European Journal of Operational Research* 151, pp. 119–139. DOI: 10.1016/S0377-2217(02)00595-7.

Ehrgott, M. and M. M. Wiecek (2005). "Mutiobjective Programming". In: Springer. Chap. Chapter 17, pp. 667–708. DOI: 10.1007/0-387-23081-8_17.

Eswaran, P., A. Ravindran, and H. Moskowitz (1989). "Algorithms for nonlinear integer bicriterion problems". In: *Journal of Optimization Theory and Applications* 63.2, pp. 261–279. DOI: 10.1007/BF00939577.

Eusébio, A. and J. Figueira (Sept. 2009a). "Finding non-dominated solutions in bi-objective integer network flow problems". In: *Computers & Operations Research* 36, 2554–2564. DOI: 10.1016/j.cor.2008.11.001.

Eusébio, A. and J. R. Figueira (2009b). "On the computation of all supported efficient solutions in multi-objective integer network flow problems". In: *European Journal of Operational Research* 199.1, 68–76. DOI: 10.1016/j.ejor.2008.10.031.

Eusébio, A., J. Figueira, and M. Ehrgott (Aug. 2014). "On finding representative non-dominated points for bi-objective integer network flow problems". In: *Computers & Operations Research* 48, 1–10. DOI: 10.1016/j.cor.2014.02.009.

Faulkenberg, S. L. and M. M. Wiecek (2010). "On the quality of discrete representations in multiple objective programming". In: *Optimization and Engineering* 11.3, pp. 423–440. DOI: 10.1007/s11081-009-9093-2.

Figueira, J. R., C. M. Fonseca, P. Halffmann, K. Klamroth, L. Paquete, S. Ruzika, B. Schulze, M. Stiglmayr, and D. Willems (2017). "Easy to say they're hard, but hard to see they're easy—Towards a categorization of tractable multiobjective combinatorial optimization problems". In: *Journal of Multi-Criteria Decision Analysis* 24.1-2, pp. 82–98. DOI: 10.1002/mcda.1574.

Fonseca, C., L. Paquete, and M. Lopez-Ibanez (2006). "An Improved dimension-sweep algorithm for the hypervolume indicator". In: *2006 IEEE International Conference on Evolutionary Computation*, pp. 1157–1163. DOI: 10.1109/CEC.2006.1688440.

Ford, L. R. and D. R. Fulkerson (1956). "Maximal Flow Through a Network". In: *Canadian Journal of Mathematics* 8, pp. 399–404. DOI: 10.4153/CJM-1956-045-5.

Ford, L. R. and D. R. Fulkerson (1962). *Flows in Networks*. Princeton, NJ: Princeton University Press.

Gal, T. (Feb. 1977). "A general method for determining the set of all efficient solutions to a linear vectormaximum problem". In: *European Journal of Operational Research* 1, 307–322. DOI: 10.1016/0377-2217(77)90063-7.

Galle, P. (Sept. 1989). "Branch & Sample : A simple strategy for constraints satisfaction". In: *BIT Numerical Mathematics* 29. DOI: 10.1007/BF02219227.

Gandibleux, X., H. Morita, and N. Katoh (2001). "The Supported Solutions Used as a Genetic Information in a Population Heuristic". In: *Evolutionary Multi-Criterion Optimization*. Ed. by E. Zitzler, L. Thiele, K. Deb, C. A. Coello Coello, and D. Corne. Vol. 1993. Lecture Notes in Computer Science. Springer, pp. 429–442. DOI: 10.1007/3-540-44719-9_30.

Gandibleux, X., H. Morita, and N. Katoh (2003). "Use of a genetic heritage for solving the assignment problem with two objectives". In: *International Conference on Evolutionary Multi-Criterion Optimization*. Springer, pp. 43–57. DOI: 10.1007/978-3-540-24777-7_5.

Gass, S. and T. Saaty (1955). "The computational algorithm for the parametric objective function". In: *Naval Research Logistics Quarterly* 2 (1-2), pp. 39–45. DOI: 10.1002/nav.3800020106.

"Generating representative sets for multiobjective discrete optimization problems with specified coverage errors" (2025). In: *Computational Optimization and Applications* 90.1, 27–51. DOI: 10.1007/s10589-024-00627-y.

Geoffrion, A. M. (1968). "Proper efficiency and the theory of vector maximization". In: *Journal of Mathematical Analysis and Applications* 22.3, 618–630. DOI: 10.1016/0022-247X(68)90201-1.

Goldberg, A. V. and R. E. Tarjan (1989). "Finding minimum-cost circulations by canceling negative cycles". In: *Proceedings of the twenty-first annual ACM symposium on Theory of computing*. ACM, pp. 388–397. DOI: 10.1145/73007.73042.

Gomory, R. E. (1958). "An algorithm for integer solutions to linear programs". In: *Recent Advances in Mathematical Programming*, pp. 269–302. DOI: 10.1090/S0002-9904-1958-10224-4.

Haimes, Y., L. S. Lasdon, and D. Wismer (1971). "On a bicriterion formation of the problems of integrated system identification and system optimization". In: *IEEE Trans-*

actions on Systems, Man, and Cybernetics, pp. 296–297. DOI: 10.1109/TSMC.1971.4308298.

Halffmann, P., L. E. Schäfer, K. Dächert, K. Klamroth, and S. Ruzika (2022). "Exact algorithms for multiobjective linear optimization problems with integer variables: A state of the art survey". In: *Journal of Multi-Criteria Decision Analysis* 29.5-6, 341–363. DOI: 10.1002/mcda.1780.

Hamacher, H. W., C. R. Pedersen, and S. Ruzika (2007a). "Finding representative systems for discrete bicriteria optimization problems by box algorithms". In: *Operations Research Letters 35* 3, pp. 336–344. DOI: 10.1016/j.orl.2006.03.019.

Hamacher, H. W. (1995). "A note on K best network flows." In: *Annals OR* 57.1, 65–72. DOI: 10.1007/BF02099691.

Hamacher, H. W., C. R. Pedersen, and S. Ruzika (2007b). "Multiple objective minimum cost flow problems: A review." In: *European Journal Operation Research* 176.3, 1404–1422. DOI: 10.1016/j.ejor.2005.09.033.

Hamacher, H. W. and G. Ruhe (1994). "On spanning tree problems with multiple objectives". In: *Annals of Operations Research* 52, pp. 209–230. DOI: 10.1007/BF02032304.

Hansen, P. (1980). "Bicriterion path problems". In: *Multiple Criteria Decision Making Theory and Application*. Ed. by G. Fandel and T. Gal. Vol. 177. Lecture Notes in Economics and Mathematical Systems. Springer, pp. 109–127. DOI: 10.1007/978-3-642-48782-8.

Harel, D. and R. E. Tarjan (1984). "Fast Algorithms for Finding Nearest Common Ancestors". In: *SIAM J. Comput.* 13.2, 338–355. DOI: 10.1137/0213024.

Herzel, A., S. Ruzika, and C. Thielen (2021). "Approximation Methods for Multiobjective Optimization Problems: A Survey". In: *INFORMS Journal on Computing* 33.4, pp. 1284–1299. DOI: 10.1287/ijoc.2020.1028.

Hopcroft, J. and R. Tarjan (1973). "Algorithm 447: Efficient algorithms for graph manipulation". In: *Communications of the ACM* 16.6, pp. 372–378. DOI: 10.1145/362248.362272.

Isermann, H. (1974). "Technical Note—Proper efficiency and the linear vector maximum problem". In: *Operations Research* 22.1, 189–191. DOI: 10.1287/opre.22.1.189.

Jakubovski Filho, H. L., T. N. Ferreira, and S. R. Vergilio (2019). "Preference based multi-objective algorithms applied to the variability testing of software product lines". In: *Journal of Systems and Software* 151, 194–209. DOI: 10.1016/j.jss.2019.02.028.

Jesus, A. D. B. d. (2015). "Implicit Enumeration for Representation Systems in Multi-objective Optimization". MA thesis. University of Coimbra.

Johnson, D. S., M. Yannakakis, and C. H. Papadimitriou (1988). "On generating all maximal independent sets". In: *Information Processing Letters* 27.3, 119–123. DOI: 10.1016/0020-0190(88)90065-8.

Jünger, M. and P. Mutzel, eds. (2004). *Graph Drawing Software*. Springer.

Kaplan, H., N. Rubin, M. Sharir, and E. Verbin (2008). "Efficient colored orthogonal range counting". In: *SIAM Journal on Computing* 38.3, pp. 982–1011. DOI: 10.1137/070684483.

Karmarkar, N. (1984). "A new polynomial-time algorithm for linear programming". In: *Combinatorica* 4.4, pp. 373–395. DOI: 10.1007/BF02579150.

Karp, R. M. (1972). "Reducibility among combinatorial problems". In: *Complexity of Computer Computations*. New York: Plenum Press, pp. 85–103. DOI: 10.1007/978-1-4684-2001-2_9.

Kellerer, H., U. Pferschy, and D. Pisinger (2004). *Knapsack Problems*. Springer, Berlin, Germany.

Kirlik, G. and S. Sayın (2014). "A new algorithm for generating all nondominated solutions of multiobjective discrete optimization problems". In: *European Journal of Operational Research* 232, pp. 479–488. DOI: 10.1016/j.ejor.2013.08.001.

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. DOI: 10.1016/j.ejor.2015.03.031.

Klein, D. and E. Hannan (1982). "An algorithm for the multiple objective integer linear programming problem". In: *European Journal of Operational Research* 9, pp. 378–385. DOI: 10.1016/0377-2217(82)90182-5.

Klein, M. (1967). "A primal method for minimal cost flows with applications to the assignment and transportation problems". In: *Management Science* 14.3, pp. 205–220. DOI: 10.1287/mnsc.14.3.205.

Klingman, D. D., A Napier, and J. D. Stutz (1974). "NETGEN: A program for generating large scale capacitated assignment, transportation, and minimum cost flow network problems". In: *Management Science* 20, pp. 814–821. DOI: 10.1287/MNSC.20.5.814.

Korte, B. H. and J. Vygen (2012). *Combinatorial Optimization: Theory and Algorithms*. New York, NY: Springer-Verlag. DOI: 10.1007/978-3-642-24488-9.

Kruskal, J. B. (1956). "On the shortest spanning subtree of a graph and the traveling salesman problem". In: *Proceedings of the American Mathematical Society* 7.1, pp. 48–50.

Könen, D. (2019). "Über das Zählen und Bestimmen aller ganzzahligen optimalen Netzwerkflüsse". Master's thesis. University of Cologne.

Könen, D., D. Schmidt, and C. Spisla (2022a). "Finding all minimum cost flows and a faster algorithm for the K best flow problem". In: *Discrete Applied Mathematics* 321, 333–349. DOI: 10.1016/j.dam.2022.07.007.

Könen, D., D. Schmidt, and C. Spisla (2022b). "Finding all minimum cost flows and a faster algorithm for the K best flow problem". In: *Discrete Applied Mathematics* 321, 333–349. DOI: 10.1016/j.dam.2022.07.007.

Könen, D. and M. Stiglmayr (2023). *Output-sensitive complexity of multi-objective integer network flow problems*. arXiv preprint, submitted to the Journal of Combinatorial Optimization, arXiv: 2312.01786 (cs.CC).

Könen, D. and M. Stiglmayr (2025a). "An output-polynomial time algorithm to determine all supported efficient solutions for multi-objective integer network flow problems". In: *Discrete Applied Mathematics* 376, 1–14. DOI: 10.1016/j.dam.2025.06.001.

Könen, D. and M. Stiglmayr (2025b). *On supportedness in multi-objective combinatorial optimization*. arXiv preprint, submitted to the Journal of Multi-Criteria Decision Analysis ,arXiv: 2501.13842 (math.OC).

Land, A. H. and A. G. Doig (1960). "An automatic method of solving discrete programming problems". In: *Econometrica* 28.3, pp. 497–520. DOI: 10.2307/1910129.

Laumanns, M., L. Thiele, and E. Zitzler (2005). "An adaptive scheme to generate the pareto front based on the epsilon-constraint method". In: *Practical Approaches to Multi-Objective Optimization*. Ed. by J. Branke, K. Deb, K. Miettinen, and R. E. Steuer. Dagstuhl Seminar Proceedings 04461. Dagstuhl, Germany: Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

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, pp. 932–942. DOI: 10.1016/j.ejor.2004.08.029.

Lawler, E. L., J. K. Lenstra, and A. H. G. Rinnooy Kan (1980). "Generating all maximal independent sets: NP-hardness and polynomial-time algorithms". In: *SIAM Journal on Computing* 9.3, 558–565. DOI: 10.1137/0209042.

Liefooghe, A., S. Verel, and J.-K. Hao (2014). "A hybrid metaheuristic for multiobjective unconstrained binary quadratic programming". In: *Applied Soft Computing* 16, pp. 10–19. DOI: 10.1016/j.asoc.2013.11.008.

Liefooghe, A., S. Verel, L. Paquete, and J.-K. Hao (2015). "Experiments on local search for bi-objective unconstrained binary quadratic programming". In: *Evolutionary Multi-Criterion Optimization: 8th International Conference, EMO 2015, Guimarães, Portugal, March 29–April 1, 2015. Proceedings, Part I 8*. Springer, pp. 171–186. DOI: 10.1007/978-3-319-15934-8_12.

Lokman, B. and M. Köksalan (2013). "Finding all nondominated points of multi-objective integer programs". In: *Journal of Global Optimization* 57 (2), pp. 347–365. DOI: 10.1007/s10898-012-9955-7.

Mavrotas, G. (2009). "Effective implementation of the ε-constraint method in Multi-Objective Mathematical Programming problems". In: *Applied Mathematics and Computation* 213, pp. 455–465. DOI: 10.1016/j.amc.2009.03.037.

Medrano, F. A. and R. L. Church (2014). "Corridor location for infrastructure development: A fast bi-objective shortest path method for approximating the pareto frontier". In: *International Regional Science Review* 37.2, 129–148. DOI: 10.1177/0160017613507772.

Medrano, F. A. and R. L. Church (2015). "A parallel computing framework for finding the supported solutions to a bi-objective network optimization problem". In: *Journal of Multi-Criteria Decision Analysis* 22.5-6, 244–259. DOI: 10.1002/mcda.1541.

Mendonça Neta, B. M. de, G. H. D. Araújo, F. G. Guimarães, R. C. Mesquita, and P. Y. Ekel (2012). "A fuzzy genetic algorithm for automatic orthogonal graph drawing." In: *Appl. Soft Comput.* 12.4, 1379–1389. DOI: 10.1016/j.asoc.2011.11.023.

Mesquita-Cunha, M., J. R. Figueira, and A. P. Barbosa-Póvoa (2023). "New epsilon-constraint methods for multi-objective integer linear programming: A Pareto front

representation approach". In: *European Journal of Operational Research* 306.1, 286–307. DOI: 10.1016/j.ejor.2022.07.044.

Miettinen, K. and M. M. Mäkelä (2002). "On scalarizing functions in multiobjective optimization". In: *OR Spectrum* 24.2. 10.1007/s00291-001-0092-9, pp. 193–213. DOI: 10.1007/s00291-001-0092-9.

Nemhauser, G. L. and L. A. Wolsey (1999). *Integer and combinatorial optimization*. eng. Wiley-Interscience series in discrete mathematics and optimization. New York, NY ; Weinheim [u.a.]: Wiley, XIV, 763 S.

Okamoto, Y. and T. Uno (2011). "A polynomial-time-delay and polynomial-space algorithm for enumeration problems in multi-criteria optimization". In: *European Journal of Operational Research* 210.1, 48–56. DOI: 10.1016/j.ejor.2010.10.008.

Orlin, J. B. (Apr. 1993). "A faster strongly polynomial minimum cost flow algorithm". In: *Operations Research* 41.2, 338–350. DOI: 10.1287/opre.41.2.338.

Orlin, J. B. (1997). "A polynomial time primal network simplex algorithm for minimum cost flows". In: *Mathematical Programming* 78.2, pp. 109–129. DOI: 10.1007/BF02614365.

Özlen, M. and M. Azizoğlu (2009). "Multi-objective integer programming: A general approach for generating all non-dominated solutions". In: *European Journal of Operations Research* 199, pp. 25–35. DOI: 10.1016/j.ejor.2008.10.023.

Özlen, M., B. A. Burton, and C. A. G. MacRae (2014). "Multi-Objective Integer Programming: An Improved Recursive Algorithm". In: *Journal of Optimization Theory and Applications* 160.2, pp. 470–482. DOI: 10.1007/s10957-013-0364-y.

Özpeynirci, O. and M. Köksalan (2010). "An exact algorithm for finding extreme supported nondominated points of multiobjective mixed integer programs". In: *Management Science* 56.12, 2302–2315. DOI: 10.1287/mnsc.1100.1248.

Özpeynirci, Ö. and M. Köksalan (2010). "Pyramidal tours and multiple objectives". In: *Journal of Global Optimization* 48, pp. 569–582. DOI: 10.1007/s10898-009-9505-0.

Papadimitriou, C. H. (1994). *Computational Complexity*. Reading, MA: Addison-Wesley.

Papadimitriou, C. H. and M. Yannakakis (1991). "Optimization, approximation, and complexity classes". In: *Journal of Computer and System Sciences* 43.3, 425–440. DOI: 10.1016/0022-0000(91)90023-X.

Paquete, L., B. Schulze, M. Stiglmayr, and A. C. Lourenço (2022). "Computing representations using hypervolume scalarizations". In: *Computers & Operations Research* 137, p. 105349. DOI: 10.1016/j.cor.2021.105349.

Pascoletti, A. and P. Serafini (1984). "Scalarizing Vector Optimization Problems". In: *Journal of Optimization Theory and Applications* 42, pp. 499–524. DOI: 10.1007/BF00934564.

Pasternak, H. and U. Passy (1973). "Bicriterion mathematical programs with boolean variables". In: *Multiple Criteria Decision Making*. Ed. by J. Cochrane and M. Zeleny. Columbia, SC: University of South Carolina Press, pp. 327–348.

Pettersson, W. and M. Ozlen (Feb. 2019a). "Multi-objective mixed integer programming: An objective space algorithm". In: *AIP Conference Proceedings* 2070.1, p. 020039. DOI: 10.1063/1.5090006.

Pettersson, W. and M. Ozlen (Sept. 2019b). "Multiobjective integer programming: synergistic parallel approaches". In: *INFORMS Journal on Computing*. DOI: 10.1287/ijoc.2018.0875.

Pisinger, D. (1999). "Linear Time Algorithms for Knapsack Problems with Bounded Weights". In: *Journal of Algorithms* 33.1, 1–14. DOI: 10.1006/jagm.1999.1034.

Prim, R. C. (1957). "Shortest connection networks and some generalizations". In: *Bell System Technical Journal* 36.6, pp. 1389–1401. DOI: 10.1002/j.1538-7305.1957.tb01515.x..

Przybylski, A. and X. Gandibleux (2017). "Multi-objective branch and bound". In: *European Journal of Operational Research* 260.3, 856–872. DOI: 10.1016/j.ejor.2017.01.032.

Przybylski, A., X. Gandibleux, and M. Ehrgott (2008). "Two phase algorithms for the biobjective assignment problem". In: *European Journal of Operational Research* 185.2, 509–533. DOI: 10.1016/j.ejor.2006.12.054.

Przybylski, A., X. Gandibleux, and M. Ehrgott (2009). "Computational Results for Four Exact Methods to Solve the Three-Objective Assignment Problem". In: *Multiobjective Programming and Goal Programming: Theoretical Results and Practical Applications*. Ed. by V. Barichard, M. Ehrgott, X. Gandibleux, and V. T'Kindt. Lecture Notes in Economics and Mathematical Systems 618. Springer, pp. 79–88. DOI: 10.1007/978-3-540-85646-7_8.

Przybylski, A., X. Gandibleux, and M. Ehrgott (Aug. 2010a). "A recursive algorithm for finding all nondominated extreme points in the outcome set of a multiobjective integer programme". In: *INFORMS Journal on Computing* 22, 371–386. DOI: 10.1287/ijoc.1090.0342.

Przybylski, A., X. Gandibleux, and M. Ehrgott (2010b). "A two phase method for multiobjective integer programming and its application to the assignment problem with three objectives". In: *Discrete Optimization* 7.3, 149–165. DOI: 10.1016/j.disopt.2010.03.005.

Przybylski, A., K. Klamroth, and R. Lacour (2019). *A simple and efficient dichotomic search algorithm for multi-objective mixed integer linear programs*. arXiv: 1911.08937 [math.OC].

Raith, A. and M. Ehrgott (June 2009). "A two-phase algorithm for the biobjective integer minimum cost flow problem". In: *http://www.esc.auckland.ac.nz/research/tech/esc-tr-661.pdf* 36. DOI: 10.1016/j.cor.2008.06.008.

Raith, A. and A. Sedeño-Noda (Jan. 2017). "Finding extreme supported solutions of biobjective network flow problems: An enhanced parametric programming approach". In: *Computers & Operations Research* 82. DOI: 10.1016/j.cor.2017.01.004.

Ralphs, T., M. Saltzman, and M. M. Wiecek (2006). "An improved algorithm for solving biobjective integer programs". In: *Annals of Operations Research* 147, pp. 43–70. DOI: 10.1007/s10479-006-0058-z.

Renegar, J. (1988). "A polynomial-time algorithm, based on Newton's method, for linear programming". In: *Mathematical Programming* 40.1, 59–93. DOI: 10.1007/BF01580724.

Ruhe, G. (1988). "Complexity results for multicriterial and parametric network flows using a pathological graph of Zadeh". In: *Zeitschrift für Operations Research* 32.1, 9–27. DOI: 10.1007/BF01920568.

Sayın, S. and P. Kouvelis (2005). "The multiobjective discrete optimization problem: A weighted min-max two-stage optimization approach and a bicriteria algorithm". In: *Management Science* 51.10, pp. 1572–1581. DOI: 10.1287/mnsc.1050.0413.

Sayın, S. (2000). "Measuring the quality of discrete representations of efficient sets in multiple objective mathematical programming". In: *Mathematical Programming* 87.3, pp. 543–560. DOI: 10.1007/PL00011306.

Sayın, S. (2024). "Supported nondominated points as a representation of the nondominated set: An empirical analysis". In: *Journal of Multi-Criteria Decision Analysis* 31.1-2, e1829. DOI: 10.1002/mcda.1829.

Schrijver, A. (2003). *Combinatorial Optimization - Polyhedra and Efficiency*. Springer.

Schrijver, A. (1998). *Theory of Linear and Integer Programming*. John Wiley & Sons.

Schulze, B. (2017). "New perspectives on multi-objective knapsack problems". PhD thesis. Universität Wuppertal, 2017.

Schulze, B., K. Klamroth, and M. Stiglmayr (2019). "Multi-objective unconstrained combinatorial optimization: a polynomial bound on the number of extreme supported solutions". In: *Journal of Global Optimization* 74.3, 495–522. DOI: 10.1007/s10898-019-00745-6.

Schutze, O., X. Esquivel, A. Lara, and C. A. C. Coello (2012). "Using the averaged Hausdorff distance as a performance measure in evolutionary multiobjective optimization". In: *IEEE Transactions on Evolutionary Computation* 16.4, pp. 504–522. DOI: 10.1109/TEVC.2011.2161872.

Sedeño-Noda, A. and J. J. Espino-Martín (2013). "On the K best integer network flows." In: *Computers & Operations Research* 40.2, 616–626. DOI: 10.1016/j.cor.2012.08.014.

Sedeño-Noda, A. and A. Raith (2015). "A Dijkstra-like method computing all extreme supported non-dominated solutions of the biobjective shortest path problem". In: *Computers & Operations Research* 57, 83–94. DOI: 10.1016/j.cor.2014.11.010.

Serafini, P. (1986). "Some Considerations About Computational Complexity for Multi-Objective Combinatorial Problems". In: *Recent Advances and Historical Development of Vector Optimization*. Vol. 294. Lecture Notes in Economics and Mathematical Systems. Springer, pp. 222–231. DOI: 10.1007/978-3-642-46618-2_15.

Silva, C. G. da and J. Clímaco (2007). "A note on the computation of supported non-dominated solutions in the bi-criteria minimum spanning tree problem". In: *Journal of Telecommunications and Information Technology*, pp. 11–15. DOI: 10.26636/jtit.2007.4.0003.

Sipser, M. (2012). *Introduction to the Theory of Computation*. 3rd. Boston, MA: Cengage Learning.

Sourd, F., O. Spanjaard, and P. Perny (2006). "Multi-objective branch and bound. Application to the bi-objective spanning tree problem". In: *7th international conference in multi-objective programming and goal programming*.

Steuer, R. E. (1986). *Multiple criteria optimization : theory, computation, and application*. eng. Wiley series in probability and mathematical statistics ; Applied probability and mathematical statistics. New York ; Wiley.

Sun, M. (July 2011). "Finding integer efficient solutions for multiple objective network programming problems". In: *Networks* 57, 362–375. DOI: 10.1002/net.20407.

Sylva, J. and A. Crema (2004). "A method for finding the set of non-dominated vectors for multiple objective integer linear programs". In: *European Journal of Operational Research* 158, pp. 46–55. DOI: 10.1016/S0377-2217(03)00255-8.

Sylva, J. and A. Crema (2007). "A method for finding well-dispersed subsets of non-dominated vectors for multiple objective mixed integer linear programs". In: *European Journal of Operational Research* 180, pp. 1011–1027. DOI: 10.1016/j.ejor.2006.02.049.

Tamby, S. (2018). "Approches génériques pour la résolution de problèmes d'optimisation discrète multiobjectif". PhD thesis. Université Paris-Dauphine.

Tamby, S. and D. Vanderpooten (2020). "Enumeration of the nondominated set of multiobjective discrete optimization problems". In: *INFORMS Journal on Computing*. DOI: 10.1287/ijoc.2020.0953.

Tarjan, R. E. (1997). "Dynamic trees as search trees via Euler tours, applied to the network simplex algorithm". In: *Mathematical Programming* 78.2, pp. 169–177. DOI: 10.1007/BF02614369.

Tenfelde-Podehl, D. (2003). *A Recursive Algorithm for Multiobjective Combinatorial Optimization Problems with q Criteria*. Tech. rep. Universität Graz/Technische Universität Graz, SFB F003-Optimierung und Kontrolle.

The Luc, D. (1995). "On the properly efficient points of nonconvex sets". In: *European Journal of Operational Research* 86.2, 332–336. DOI: 10.1016/0377-2217(94)00110-X.

Turgut, O., E. Dalkiran, and A. Murat (Apr. 2019). "An exact parallel objective space decomposition algorithm for solving multi-objective integer programming problems". In: *Journal of Global Optimization* 75. DOI: 10.1007/s10898-019-00778-x.

Tuyttens, D., J. Teghem, P. Fortemps, and K. V. Nieuwenhuyze (2000). "Performance of the MOSA method for the bicriteria assignment problem". In: *Journal of Heuristics* 6, pp. 295–310. DOI: 10.1023/A:1009670112978.

Ulungu, E. L. and J. Teghem (1995). "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. DOI: 10.1007/BF01096434.

Vaz, D., L. Paquete, C. M. Fonseca, K. Klamroth, and M. Stiglmayr (2015). "Representation of the non-dominated set in biobjective combinatorial optimization". In: *Computers & Operations Research* 63, pp. 172 –186. DOI: 10.1016/j.cor.2015.05.003.

Visée, M., J. Teghem, M. Pirlot, and E. Ulungu (1998). "Two-phases method and branch and bound procedures to solve the bi–objective knapsack problem". In: *Journal of Global Optimization* 12.2, 139–155. DOI: 10.1023/A:1008258310679.

Welch, J. T. (Apr. 1966). "A Mechanical Analysis of the Cyclic Structure of Undirected Linear Graphs". In: *Journal of the Association for Computing Machinery* 13.2, 205–210. DOI: 10.1145/321328.321331.

Wolsey, L. A. (1998). *Integer Programming*. John Wiley & Sons.

Yan, S. and Y.-P. Tu (2002). "A network model for airline cabin crew scheduling." In: *Eur. J. Oper. Res.* 140.3, 531–540. DOI: 10.1016/S0377-2217(01)00215-6.

Yuf, P. L. and M. Zeleny (1976). "Linear multiparametric programming by multicriteria simplex method". In: *Management Science* 23.2, 159–170. DOI: 10.1287/mnsc.23.2.159.

Zitzler, E., L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. Da Fonseca (2003). "Performance assessment of multiobjective optimizers: An analysis and review". In: *IEEE Transactions on Evolutionary Computation* 7.2, pp. 117–132. DOI: 10.1109/TEVC.2003.810758.