BERGISCHE
UNIVERSITÄT
WUPPERTAL

On Improvements of Multi-objective Branch and Bound

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

vorgelegt von
Julius C. Bauß

Erstgutachter: PD Dr. Michael Stiglmayr
Zweitgutachterin: Prof. Dr. Sophie N. Parragh

Wuppertal, Februar 2024

Acknowledgements

During my work on this dissertation I have been supported by numerous people, who always encouraged me and inspired me to develop new ideas. Time flew by but I am grateful for all the memorable moments, awesome trips and great teamwork.

First of all I would like to thank my supervisor Michael Stiglmayr for always supporting me and providing new solution approaches whenever I faced any difficulties. Especially, I thank you for your excessive LaTeX affinity, which might slightly rubbed off on me.

Moreover, I would like to thank all former and current members of the working group *Optimization* of the University of Wuppertal for the friendly atmosphere and wonderful time. I can proudly say we are more than just colleagues. I also would like to thank my co-authors Sophie Parragh and Michael Stiglmayr for the great and productive work.

Furthermore, I had the opportunity to spend some time abroad to collaborate with experts on multi-objective branch and bound. I thank Sune Lauth Gadegaard for welcoming me in Aarhus a couple of days in February 2020 and I also thank Sophie Parragh for hosting me in Linz in April 2022 and April 2023 for a week each. I was able to get new inputs and further research ideas during all these trips.

A special thanks to Nicolas Forget for providing me with a baseline implementation on which I could base my code on.

Additionally, I also like to thank Deutsche Forschungsgemeinschaft (project number KL 1076/11-1) for the partial financial support.

Last but not least I thank my family and friends for always supporting and encouraging me whenever problems occurred. I particularly thank my parents, Kerstin and Achim Bauß. Thank you for your love, care and support.

Contents

1 Introduction

In today's profit-oriented world, most of the everyday decisions, especially the ones with an economic context, can be modeled as a mathematical optimization problem. Imagine the following situation: a company has to decide which projects should be realized in the next quarter. For every possible project the expected costs and the expected revenue are known. Since the company has a limited budget, it is not possible to realize all projects, i.e., it is necessary to select a subset of projects. Of course, the manager aims to find a set of projects respecting the budget that maximizes the expected profit. This kind of problem belongs to the so-called single-objective integer optimization models. The name derives from the integrality of the decision variables. In the considered project selection problem, the variables are not just integer but binary, which is a special case of integer problems. Each variable models the decision if the corresponding should be realized or not. Integer optimization problems are, in general, harder to solve than their continuous counterparts. A common method to solve these kinds of problems is the *branch and bound approach*. This algorithm divides an underlying problem, which is too hard to be solved directly, into easier subproblems. In the last 50 years, the popularity of integer programs arose and therefore more and more approaches have been developed (Nemhauser and Wolsey, 1988). Hence, commercial single objective solvers, based on the branch and bound framework gained popularity. The performance of single-objective solvers like CPLEX and Gurobi has improved tremendously by the factor of about "100 000" in the last 30 years (Bixby, 2012). Thus, these commercial solvers and therewith branch and bound algorithms are the gold-standard for single-objective integer optimization problems.

Now recall the project selection problem: the manager now wants to additionally consider the sustainability-index of each project. By maximizing this index a second objective is added to the previous problem. Thus, the problem transforms into a bi-objective integer optimization problem. By adding even more objectives it transforms into a multi-objective problem. Since these objectives are conflicting there is, in general, no solution which optimizes both objectives simultaneously. Hence, compromise solutions have to be found where it is not possible to improve one of the objectives without worsening at least one of the others. Those solutions are called *efficient* or *Pareto-optimal*. The underlying optimality concept has been derived by Vilfredo Pareto (1848–1923) and Francis Edgeworth (1845–1926) in the late 1800s and is the most common in multi-objective optimization. While branch and bound based algorithms are a standard approach to solve single-objective inte-

ger optimization problems, multi-objective branch and bound methods are rarely applied compared to the predominant objective space methods. This is due to two reasons. On the one hand, objective space methods rely on solving a series of scalarized single-objective subproblems. Therefore, they benefit from the huge improvements of the optimized single-objective solvers like CPLEX or Gurobi (see Bixby, 2012). On the other hand, branch and bound approaches suffer from considerably weaker bounding in multiple objectives. However, objective space methods always solve an integer problem from scratch. That is one of the reasons why there is an increasing research interest on decision space methods in the last decades, in particular on the branch and bound method.

A multi-objective branch and bound operates in the same way as its well-known single-objective version. Since the considered problem is too hard to be solved directly, it is divided into easier subproblems. Every created subproblem is associated with a node in a tree data structure. Thereby, a node i is the child node of node j if and only if the feasible set of the subproblem corresponding to node i is a subset of the feasible set of the (sub)problem corresponding to node j. One of the first multi-objective branch and bound methods with an underlying tree structure was proposed in Klein and Hannan (1982). In each iteration an active node is selected and its corresponding lower bound set is computed. The algorithm is initialized by investigating the root node to which the original problem is associated. After the computation of the lower bound set, the upper bound set is possibly updated and it is checked if it the node can be fathomed. If the node cannot be fathomed, the corresponding problem has to be divided further into new subproblems.

1.1 Outline of this Thesis

Branch and bound approaches are, as stated above, rarely used compared to objective space methods in the multi-objective case. We propose improvements to increase the performance of multi-objective branch and bound by using objective space information to limit the impact of the mentioned shortcomings.

This thesis is organized in seven chapters. In the first three chapters, we introduce basic mathematical concepts and present an extensive overview of solution methods. In the remaining chapters, new strategies and approaches are proposed to improve the performance of multi-objective branch and bound methods. We start by augmenting in the bi-objective case and transfer those results and approaches to the multi-objective case. Finally, different combinations of certain key components of branch and bound algorithms are compared. In the following the content of each chapter is described in more detail.

Chapter 2 Basic concepts and definitions which are relevant in the remainder of this work are presented. We start by introducing multi-objective optimization models with their different properties. Since there is no natural order of the $\mathbb{R}^p$, different dominance relations used in multi-objective optimization are summarized and corresponding optimality conditions are defined. Afterwards, we give description of different upper and lower bound sets, which are one of the key components of multi-objective branch and bound frameworks. Moreover, a brief overview of polyhedral theory, which is mainly needed for the description and computation of certain lower bound sets, is given.

Chapter 3 A detailed overview of different solution methods is provided. These are distinguished between the so-called *objective space methods* and *decision space methods*. Objective space methods scalarize the underlying problem by replacing it by a series of single-objective problems. We give detailed descriptions of the most common scalarization techniques, discuss their advantages and shortcomings, review related work and present bi-objective examples. In the second part of this chapter, decision space methods, in particular dynamic programming and the branch and bound algorithm, are discussed. We describe its key components and present different approaches and techniques which are proposed by literature.

Chapter 4 We focus on bi-objective integer optimization problems and therefore present bi-objective branch and bound algorithms that are augmented by scalarization-based information. A new node selection rule is introduced based on the approximated hypervolume gap between lower and upper bound. Additionally, approaches to improve the bound sets which lead to a higher probability to fathom a node by dominance are presented. Their tremendous impact on the computational performances is evaluated regarding different problem classes, namely knapsack, uncapacitated facility location and assignment problems.

Chapter 5 The approaches, presented in Chapter 4, are extended to the multi-objective case with three and more objectives. Thereby, additional difficulties have to be considered since some of the properties used in Chapter 4 only hold in the bi-objective case. Furthermore, we introduce another gap measure between lower and upper bound since the approximated hypervolume gap strategy might be too costly for an increasing number of objectives. Additionally, new approaches using objective space information to improve the bounds by, e.g., warm starting the upper and lower bound set are proposed. The corresponding remarkable impact on the performance of multi-objective branch and bound is tested on multi-objective benchmark instances which include knapsack, uncapacitated

facility location, capacitated facility location and generalized assignment problems.

Chapter 6 The previous chapters showed that the order in which the subproblems/nodes are considered is crucial for the performance of the branch and bound algorithm. Since there is a close correlation of the order and the total number of explored nodes, the impact of different node selection strategies combined with various branching rules is investigated. Regarding the node selection we focus on methods which rely on different gap measures. All combinations are tested on multi-objective benchmark instances which include knapsack, uncapacitated facility location and generalized assignment problems.

Chapter 7 This work is concluded by summarizing the main results.

1.2 Publications

The majority of the content of this thesis has been published or submitted to scientific journals:

- J. Bauß and M. Stiglmayr (2024b). "Augmenting bi-objective branch and bound by scalarization-based information". In: *Mathematical Methods of Operations Research*. DOI: 10.1007/s00186-024-00854-3
- J. Bauß et al. (2023). "Adaptive improvements of multi-objective branch and bound". *Submitted to EURO Journal on Computational Optimization*. DOI: 10.48550/arXiv.2312.12192
- J. Bauß and M. Stiglmayr (2024a). "Adapting branching and queuing for multi-objective branch and bound". In: *Operations Research Proceedings 2023*. Accepted for publictation. Springer. DOI: 10.48550/arXiv.2311.05980

This work contains parts from the articles Bauß and Stiglmayr (2024b), Bauß et al. (2023) and Bauß and Stiglmayr (2024a). Chapter 2 and Chapter 3 contain parts of all three mentioned articles. Chapter 4 is mainly based on Bauß and Stiglmayr (2024b), Chapter 5 is mainly based on Bauß et al. (2023) and Chapter 6 extends the article Bauß and Stiglmayr (2024a). Additionally, the Julia implementations and created test instances of the presented algorithms and approaches are publicly available in the following Git repositories:

- J. Bauß and M. Stiglmayr (2023b). *Augmented bi-objective branch and bound framework*. Git repository. URL: https://git.uni-wuppertal.de/bauss/augmented-bi-objective-branch-and-bound

- J. Bauß and M. Stiglmayr (2023a). *Adaptive multi-objective branch and bound framework*. Git repository. URL: `https://git.uni-wuppertal.de/bauss/adaptive-improvements-of-multi-objective-branch-and-bound`
- J. Bauß and M. Stiglmayr (2023c). *GAP and CFLP test instances*. Git repository. URL: `https://git.uni-wuppertal.de/bauss/generalized-assignment-problem-test-instances`

2 Multi-objective Optimization Models and Properties

In this chapter, the basic concepts, notations and definitions of multi-objective optimization are introduced. Among other things, optimality conditions are defined, the concept of bound sets is explained and basic polyhedral theory is introduced. Most of these concepts can be found in textbooks on multi-objective optimization, e.g., in Ehrgott (2005), Miettinen (1998) or Steuer (1986).

2.1 Multi-objective Optimization Models

A generic *multi-objective program* can be written in the form:

$$\begin{aligned} \min \ & \big(z_1(x), \ldots, z_p(x)\big)^\top \\ \text{s.t.} \ & x \in X, \end{aligned} \tag{MOP}$$

where $p \geq 2$ is the number of objective functions. Often the objective functions are given in form of a vector, denoted as $z(x) \coloneqq (z_1(x), \ldots, z_p(x))^\top$. A vector $x \in \mathbb{R}^n$ is called a *solution*. If additionally $x \in X$, then x is a *feasible solution*. The feasible set X is a subset of the so-called *decision space* $\mathbb{R}^n$. The image of X is defined by $Y \coloneqq \{z(x) \colon x \in X\}$ and is a subset of the *objective space* $\mathbb{R}^p$. An objective vector $y \in Y$ is called *feasible point*. In the following we will specifically consider problems where the objective functions and constraints are linear and x is restricted to be integer and $x \geqq 0$, namely a so-called *multi-objective integer linear program*. Note that "$\geqq$" is used as componentwise greater or equal condition, i.e., $x_i \geq 0$ for all $i = 1, \ldots, p$. A multi-objective linear program can be written in the following form:

$$\begin{aligned} \min \ & z(x) = C \cdot x \\ \text{s.t.} \ & A\,x \leqq b \\ & x \geqq 0 \\ & x \in \mathbb{Z}^n. \end{aligned} \tag{MOILP}$$

The objective vector $z(x)$ can be written as $z(x) = C \cdot x \in \mathbb{R}^p$ with the matrix of objective coefficients $C \in \mathbb{R}^{p \times n}$. The set of feasible solutions is denoted by $X \coloneqq \{x \in \mathbb{Z}^n : Ax \leqq b, x \geqq 0\}$, where $A \in \mathbb{R}^{m \times n}$ is the matrix of constraint coefficients. Hence, Y is given by

$Y \coloneqq \{C\,x \colon x \in X\}$. By restricting x to be binary, the problem becomes a *multi-objective 0-1 linear program*:

$$\begin{aligned} \min \quad & z(x) = \big(z_1(x), \ldots, z_p(x)\big)^\top \\ \text{s.t.} \quad & A\,x \leqq b \\ & x \in \{0,1\}^n. \end{aligned} \tag{MO01LP}$$

MO01LPs which are based on the selection of elements/items are equipped with a specific structure. This kind of problem is denoted as *multi-objective combinatorial optimization problem* (MOCO). Thereby, the selection of items is formulated using the binary indicator variables $x_i,\ i = 1, \ldots, n$, of a predefined ground set, where $x_i = 1$ iff element i is selected in the solution. This problem class contains, e.g., knapsack, assignment and facility location problems, to mention just a few. A comprehensive introduction to combinatorial optimization in multiple objectives is given in Ehrgott and Gandibleux (2000).

2.2 Optimality Conditions

While in the single-objective case ($p = 1$) feasible solutions can be compared by the value of the objective function, in the multi-objective case ($p \geq 2$) each solution is associated with an objective vector in the objective space $\mathbb{R}^p$. So in the optimization process, it is necessary to compare vectors, instead of real values. Since the $\mathbb{R}^p$ has no "natural" order, optimization is subject to a specific dominance relation. In this work we use the *Pareto concept of optimality* (see, e.g., Ehrgott, 2005), which is based on the componentwise order.

Definition 2.1. *Let $y^1, y^2 \in \mathbb{R}^p$. Then the dominance relations are defined as follows:*

- $y^1 \leqq y^2$*: y^1 weakly dominates y^2, if $y^1_j \leq y^2_j$ for $j = 1, ..., p$,*
- $y^1 < y^2$*: y^1 strictly dominates y^2, if $y^1_j < y^2_j$ for $j = 1, ..., p$,*
- $y^1 \leq y^2$*: y^1 dominates y^2, if $y^1 \leqq y^2$ and $y^1 \neq y^2$.*

Another frequently used ordering relation is the *lexicographic order*. Let $y^1, y^2 \in \mathbb{R}^p$, then the relation "$\leq_{lex}$" is defined as

$$y^1 \leq_{lex} y^2 :\Leftrightarrow y^1 = y^2 \text{ or } y^1_k < y^2_k \text{ for } k = \min\{j \in \{1, \ldots, p\} \colon y^1_j \neq y^2_j\}.$$

Definition 2.2. *Let $\pi = (\pi_1, \dots, \pi_p)$ be a permutation of the indices $1, \dots, p$. A feasible solution $\bar{x} \in X$ is called lexicographically optimal with respect to π if every solution $x \in X$ satisfies*

$$z^\pi(\bar{x}) \leq_{lex} z^\pi(x) \text{ with } z^\pi(x) = (z_{\pi_1}(x), \dots z_{\pi_p}(x)).$$

Since the objectives of an (MOP) are usually conflicting, it is, in general, impossible to find a feasible solution optimizing all objectives simultaneously. Hence, the goal is to find feasible solutions that are the best possible compromise solutions regarding all objective functions. A feasible solution $x \in X$ is *efficient* if there is no other feasible solution $\bar{x} \in X$ that dominates it ($z(\bar{x}) \leq z(x)$). In other words, for an efficient solution it is not possible to improve one of the objectives without worsening at least one of the others. Furthermore, a feasible solution $x \in X$ is *weakly efficient* if there is no other $\bar{x} \in X$ that strictly dominates it ($z(\bar{x}) < z(x)$). This allows us to define the following sets.

Definition 2.3. *The set of all efficient solutions is defined by*

$$X_E \coloneqq \{x \in X \colon \nexists \bar{x} \in X, z(\bar{x}) \leq z(x)\}.$$

Its corresponding set of images in the objective space, the non-dominated set, is defined as

$$Y_N \coloneqq \{z(x) \in Y \colon x \in X_E\}.$$

A point $y \in Y_N$ is called non-dominated point or outcome vector.

Moreover, for any set $Q \subseteq \mathbb{R}^p$ we denote by Q_N the set of its non-dominated points (i.e., $q \in Q_N \iff \nexists \bar{q} \in Q \colon \bar{q} \leq q$). Analogously, it is possible to define the set of *weakly efficient* solutions and the set of *weakly non-dominated* points.

Definition 2.4. *The set of all weakly efficient solutions is defined by*

$$X_{WE} \coloneqq \{x \in X \colon \nexists \bar{x} \in X, z(\bar{x}) < z(x)\}.$$

Its corresponding image in the objective space is defined by

$$Y_{WN} \coloneqq \{z(x) \in Y \colon x \in X_{WE}\}.$$

A point $y \in Y_{WN}$ is called weakly non-dominated.

To give further definitions of the different sets of solutions, we define some additional operators to simplify the notation. Given two sets $A, B \subseteq \mathbb{R}^p$, then $A + B$ is defined as $A + B \coloneqq \{a + b \colon a \in A, b \in B\}$. This is often referred to as the *Minkowski sum.*

Furthermore, given a set $A \subseteq \mathbb{R}^p$, a point $y \in \mathbb{R}^p$ is a *convex combination* of points of A, if there exists a finite set of points $\{y^i\}_{i=1}^t$ in A and $\lambda \in \mathbb{R}^t$ with $\lambda_i > 0,\ i = 1, \dots, t$, $\sum_{i=1}^t \lambda_i = 1$ and $y = \sum_{i=1}^t \lambda_i y^i$. The *convex hull* of A, denoted by $\text{conv}(A)$, is the set of all convex combinations of points in A (see Nemhauser and Wolsey, 1988).

The set of non-dominated points Y_N can be decomposed into two different subsets.

Definition 2.5. *The set of supported non-dominated points is defined by*

$$Y_{SN} \coloneqq \{y \in Y_N \colon y \in (\text{conv}(Y) + \mathbb{R}^p_{\geqq})_N\},$$

with $\mathbb{R}^p_{\geqq} \coloneqq \{y \in \mathbb{R}^p \colon y \geqq 0\}$. *The set*

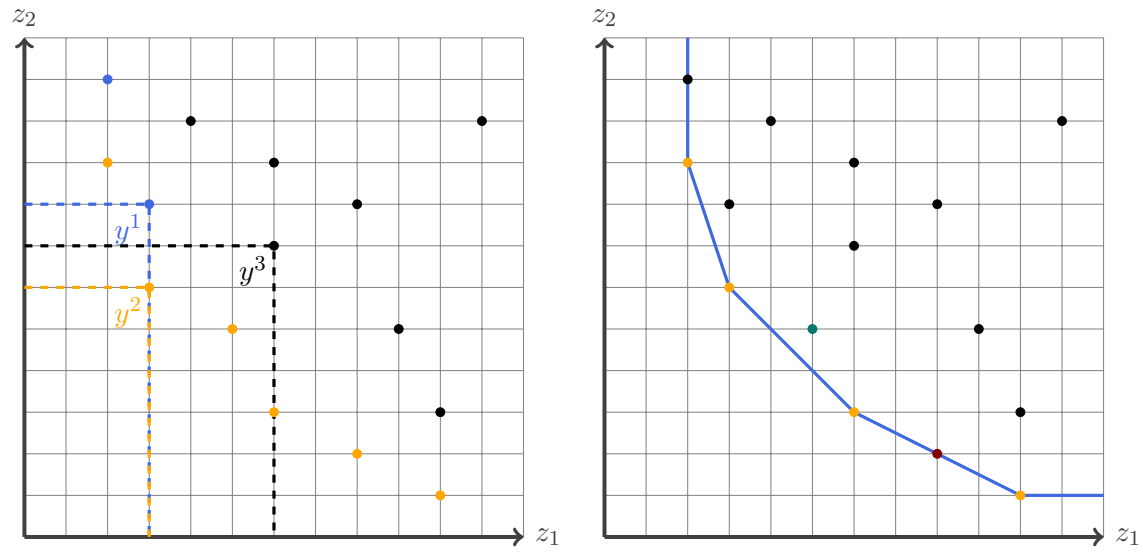
$$Y_{UN} \coloneqq \{y \in Y_N \colon y \notin Y_{SN}\}$$

is the set of all unsupported non-dominated points.

Note that the supported non-dominated points are located on the boundary of the convex hull of $Y + \mathbb{R}^p_{\geqq}$, while the unsupported non-dominated points can be located in its (relative) interior. Moreover, we distinguish the *supported non-dominated extreme* points Y_{SN1}, which are extreme points of $\text{conv}(Y) + \mathbb{R}^p_{\geqq}$ and the *supported non-dominated non-extreme* points $Y_{SN2} \coloneqq \{y \in Y_{SN} \colon y \notin Y_{SN1}\}$.

The Figure 2.1(a) shows an example of a set Y and its dominance relations. We can see that the point y^3 is (strictly) dominated by, e.g., y^2. Further, all black points are strictly dominated by at least one other point of Y. All yellow points, including y^2, are non-dominated, i.e., there are no other points that dominate them. The point y^2 dominates y^1, however not strictly. All blue points are weakly non-dominated, i.e., there is no other point in Y that strictly dominates them. Figure 2.1(b) illustrates the different subsets of the set of non-dominated points Y_N. The blue line represents the boundary of $\text{conv}(Y) + \mathbb{R}^p_{\geqq}$. The red and yellow points are supported, since they are in $(\text{conv}(Y) + \mathbb{R}^p_{\geqq})_N$. The green point is an unsupported non-dominated point. Additionally, the yellow points are supported non-dominated extreme points, i.e., they are elements of Y_{SN1}. The red point is not an extreme point and therefore an element of Y_{SN2}.

In this work we focus on solving (MO01LP), i.e., a special case of (MOILP). Nevertheless, all presented approaches can be easily transferred and used to solve (MOILP). As solution of a multi-objective 0-1 linear program we consider a *minimal complete set*, which is defined as the set of all non-dominated points $y \in Y_N$ and one corresponding efficient solution $\bar{x}$ for each non-dominated point $z(\bar{x}) = y \in Y_N$. A detailed comparison of different solution concepts is given in Serafini (1987).



(a) The yellow points in the objective space are non-dominated, while the blue points are just weakly non-dominated. The remaining black points are dominated.

(b) The yellow points are supported non-dominated extreme points, while the red point is a supported non-dominated non-extreme point. The green point is an unsupported non-dominated point.

Figure 2.1: An example of a set Y of a (MO01LP) with two objectives to show dominance relations and distinctions of different subsets of Y_N.

2.3 Bound Sets

Several solution methods use bounding techniques to limit the area that contains all non-dominated points. In the single-objective optimization, there is at most one optimal objective value $\bar{y}$ for each problem. Thus, the definition of a lower and an upper bound is straightforward.

Definition 2.6. *Let $\bar{y}$ be the optimal value of a single-objective problem. A lower and upper bound l and u on $\bar{y}$ are real values with $l \leq \bar{y} \leq u$.*

In the multi-objective case, we need to define bounds for the set of non-dominated points Y_N. Using only single points as bounds, the following points are the most popular choices.

Definition 2.7. *Let $Y \in \mathbb{R}^p$ be a set of points and $Y_N \subseteq Y$ the set of its non-dominated points. Then, the ideal point y^I, the anti-ideal point y^{AI} and the Nadir point y^N are given by:*

$$y_j^I = \min_{y \in Y} y_j, \quad y_j^{AI} = \max_{y \in Y} y_j \quad \text{and} \quad y_j^N = \max_{y \in Y_N} y_j \qquad \text{for } j = 1, \ldots, p.$$

Obviously, $y^I \leqq \bar{y} \leqq y^N \leqq y^{AI}$ holds for all $\bar{y} \in Y_N$. Furthermore, it is easy to see that y^I and y^N are the tightest upper and lower bounds of the non-dominated set Y_N consisting of single points. Although these bounds are a direct extension of the single-objective case, they are rather weak in the multi-objective case. Due to the conflicting objective functions, there is in general no optimal solution which optimizes all objectives simultaneously, which implies $y^I \neq y^N$. Hence, in general the ideal and the Nadir point can be located far away from the non-dominated points. In Figure 2.2(a), an illustration of the ideal, anti-ideal and Nadir point is given for a bi-objective example. The point y^I can be computed at the cost of solving p single-objective problems. However, the computation of y^N is in general hard for problems with three or more objectives even if the single-objective problems are solvable in polynomial time, since it corresponds to an optimization over the efficient set (see, e.g., Ehrgott and Tenfelde-Podehl, 2003; Kirlik and Sayın, 2015; Köksalan and Lokman, 2015).

In Ehrgott and Gandibleux (2001), *bound sets* are introduced that consist of a certain set of points instead of just a single point.

Definition 2.8 (cf. Ehrgott and Gandibleux, 2001)**.** *A set $\mathcal{L} \subset \mathbb{R}^p$ is a lower bound set for $\bar{Y} \subseteq Y$ if*

i) for each $y \in \bar{Y}$ there is some $l \in \mathcal{L}$ with $l \leqq y$,

ii) there is no pair $y \in \bar{Y}$ and $l \in \mathcal{L}$ with $y \leqq l$.

A set $\mathcal{U} \subset \mathbb{R}^p$ is an upper bound set for $\bar{Y} \subseteq Y$ if

i) for each $y \in \bar{Y}$ there is some $u \in \mathcal{U}$ with $y \leqq u$,

ii) there is no pair $y \in \bar{Y}$ and $u \in \mathcal{U}$ with $u \leqq y$.

In Ehrgott and Gandibleux (2007), the main results of Ehrgott and Gandibleux (2001) have been discussed in more detail. To introduce the new definition of bound sets that the authors proposed, it is necessary to define *externally stable* sets.

Definition 2.9 (cf. Ehrgott, 2005)**.** *A set $\bar{Y}$ is said to be externally stable, if* $\bar{Y} \subset (\bar{Y} + \mathbb{R}^p_{\leqq})_N$.

The following definition for bound sets ist the one this work will rely on.

Definition 2.10 (cf. Ehrgott and Gandibleux, 2007)**.** *A lower bound set $\mathcal{L} \subset \mathbb{R}^p$ for $\bar{Y} \subseteq Y$ is a*

i) $\mathbb{R}^p_{\leqq}$-closed (i.e., the set $\mathcal{L} + \mathbb{R}^p_{\leqq}$ is closed),

ii) $\mathbb{R}^p_{\leqq}$*-bounded (i.e., there exists a* $y \in \mathbb{R}^p$ *such that* $\mathcal{L} \subset y + \mathbb{R}^p_{\leqq}$*)*

iii) externally stable set (i.e., $\mathcal{L} \subset (\mathcal{L} + \mathbb{R}^p_{\leqq})_N$*),*

such that $\bar{Y} \subset (\mathcal{L} + \mathbb{R}^p_{\leqq})$*. An upper bound set* $\mathcal{U} \subset \mathbb{R}^p$ *for* $\bar{Y} \subseteq Y$ *is a*

i) $\mathbb{R}^p_{\leqq}$*-closed,*

ii) $\mathbb{R}^p_{\leqq}$*-bounded,*

iii) externally stable set,

such that $\bar{Y} \subset \mathrm{cl}\big((\mathcal{U} + \mathbb{R}^p_{\leqq})^{\complement}\big)$.

Note that although this definition was introduced for (MOCO) it also can be applied to (MOILP).

We say a lower bound set $\mathcal{L}$ is weakly dominated by an upper bound set $\mathcal{U}$ if for all $l \in \mathcal{L}$ there exists an $u \in \mathcal{U}$ such that $u \leqq l$. In the following, we use the term bound set and bound synonymously for multi-objective optimization problems. We also denote a lower bound set $\mathcal{L}$ as *convex lower bound set* or *convex lower bound* if $\mathcal{L} + \mathbb{R}^p_{\leqq}$ is a convex set. The set $\mathcal{L}$ is thereby not necessarily convex.

The standard approach to obtain lower bound sets for multi-objective integer programs is solving relaxations.

Definition 2.11 (cf. Nemhauser and Wolsey, 1988)**.** *A relaxation of an* (MOILP) *is given by any problem*

$$\begin{aligned} \min \ & \big(\bar{z}_1(x), \ldots, \bar{z}_p(x)\big)^\top \\ \text{s.t.} \ & x \in \bar{X}, \end{aligned}$$

with the following properties:

i) $X \subseteq \bar{X}$

ii) $\bar{z}(x) \leqq z(x)$ *for* $x \in X$.

The common way to obtain a lower bound for single-objective integer programs is solving its *linear relaxation*. This approach can be extended to the multi-objective case. By solving the linear relaxation of (MOILP) we obtain a valid lower bound that suits Definition 2.10 (Ehrgott and Gandibleux, 2007). The linear relaxation of a (MOILP) is formulated by relaxing its integer variable constraints.

Definition 2.12. *The linear relaxation of* (MOILP) *is given by:*

$$\begin{aligned} \min \ & \big(z_1(x), \ldots, z_p(x)\big)^\top \\ \text{s.t.} \ & A\,x \leqq b \\ & x \geqq 0 \\ & x \in \mathbb{R}^n. \end{aligned}$$

The linear relaxation of a (MO01LP) is constructed analogously by replacing the constraints $x \in \{0,1\}^n$ with $0 \leq x_i \leq 1$ for all $i = 1, \ldots, n$. Note that we add the constraints $x_i \leq 1, i = 1, \ldots, n$ to A and b in some formulations in Chapter 3, resulting in an extended constraint matrix $\bar{A}\,x \leqq \bar{b}$. In the implementation, however, variable bounds are handled implicitly. Since the set of feasible solutions for the linear relaxation is a polyhedron in the decision space and the objective functions are linear, its corresponding image in the objective space is a polyhedron as well. By only considering the non-dominated part of this polyhedron in the objective space we get a valid lower bound set for the non-dominated set of (MO01LP). A visualization of a bi-objective example is given in Figure 2.2(b).

A different valid lower bound set is obtained by solving a *convex relaxation*. This is done by computing the convex hull of the feasible points, i.e., computing $\text{conv}(Y)$. By just considering its non-dominated points (i.e. $\text{conv}(Y)_N$) we obtain the lower bound set. An illustration of a bi-objective example can be found in Figure 2.2(c). Solving a linear or convex relaxation can be done by using a dichotomic scheme (see, e.g., Aneja and Nair, 1979; Özpeynirci and Köksalan, 2010; Przybylski et al., 2010a). A detailed description of a dichotomic search algorithm is given in Section 3.1.

The classical upper bound for a single-objective integer program is given by the so-called *incumbent solution*. The incumbent solution is the currently best known solution of a problem. This concept can be easily extended to the multi-objective case by considering an *incumbent list* $X_{\mathcal{U}}$. This list contains all integer feasible solutions whose corresponding outcome vectors are not dominated by other feasible solutions found so far. An upper bound $\mathcal{U}$ is given by the image of the incumbent list, i.e., $\mathcal{U} \coloneqq z(X_{\mathcal{U}})$. Based on the upper bound set $\mathcal{U}$ we can determine the set of so-called *local upper bounds* which bound the area where new non-dominated points can be located. Note that in the literature these points are also denoted as *local Nadir points* or *corner points*. Let $\mathcal{U}$ be the upper bound set given by the incumbent list $X_{\mathcal{U}}$. Then we denote the set of local upper bounds by $\mathcal{D}(\mathcal{U})$. In Klamroth et al. (2015), a formal definition for multiple objectives is given. Furthermore, algorithms to compute and update the set of local upper bounds are presented. We slightly adapt the definitions, proposed by Klamroth et al. (2015), to our notation and the previous definition of bound sets. Let $u \in \mathcal{D}(\mathcal{U})$ be a local upper bound. Then a *search zone* $C(u)$

is defined by $C(u) \coloneqq \{y \in Y : y \leqq u\}$.

Definition 2.13 (cf. Klamroth et al., 2015)**.** *Let $X_{\mathcal{U}}$ be an incumbent list and $\mathcal{U} \coloneqq z(X_{\mathcal{U}})$. Then the corresponding set of local upper bounds $\mathcal{D}(\mathcal{U})$ is called an upper bound set with respect to $\mathcal{U}$ if and only if*

i) $\operatorname{cl}\big((\mathcal{U} + \mathbb{R}^p_{\leqq})^{\complement}\big) = \bigcup_{u \in \mathcal{D}(\mathcal{U})} C(u)$,

ii) $\forall u^1, u^2 \in \mathcal{D}(\mathcal{U}), u^1 \neq u^2, C(u^1) \not\subset C(u^2)$.

For the bi-objective case, the formal definition of the set of local upper bounds is given in the following.

Definition 2.14 (cf. Klamroth et al., 2015)**.** *Let $\mathcal{U} = \{(y_1^1, y_2^1)^\top, \dots, (y_1^t, y_2^t)^\top\}$ be an externally stable set with $t > 1$. Since an externally stable set in $\mathbb{R}^2$ can be naturally ordered, we assume $y_1^1 < \dots < y_1^t$ and $y_2^t < \dots < y_2^1$. Let $\bar{M} = (M, M)^\top \in \mathbb{R}^2$ be the global upper bound with $y < \bar{M}$ for all $y \in Y$. The set of local upper bounds is then given by:*

$$\mathcal{D}(\mathcal{U}) \coloneqq \{(y_1^1, M)^\top, (y_1^2, y_2^1)^\top, (y_1^3, y_2^2)^\top, \dots, (y_1^n, y_2^{n-1})^\top, (M, y_2^n)^\top\}.$$

In Figure 2.2(d), a set of local upper bounds is visualized for a bi-objective example.

2.4 Polyhedral Theory

In this section, a brief overview of *polyhedral theory* is given. For a more detailed description, we refer to, e.g., Nemhauser and Wolsey (1988) or Schrijver (2003).

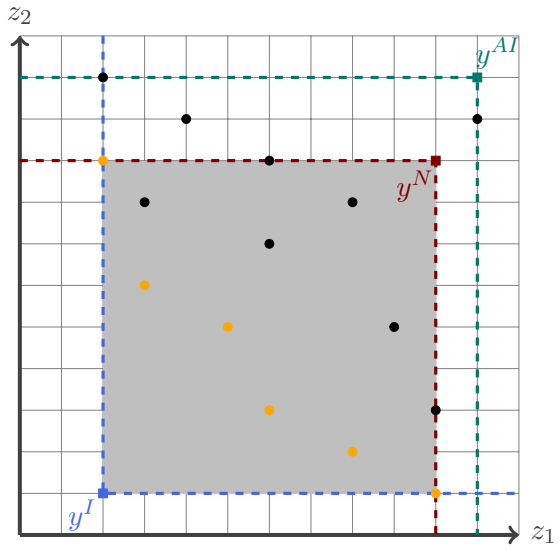
A *hyperplane* $\mathcal{H}$ is defined by $\mathcal{H} \coloneqq \{y \in \mathbb{R}^p \colon a^\top y = a_0\}$ with its corresponding normal vector $a \in \mathbb{R}^p$. A hyperplane separates the space $\mathbb{R}^p$ into the two *half-spaces* $\mathcal{H}^+ \coloneqq \{y \in \mathbb{R}^p \colon a^\top y \geq a_0\}$ and $\mathcal{H}^- \coloneqq \{y \in \mathbb{R}^p \colon a^\top y \leq a_0\}$.

Definition 2.15 (cf. Nemhauser and Wolsey, 1988)**.** *A polyhedron $\mathcal{P} \subseteq \mathbb{R}^p$ is the set of points that satisfy a finite number of inequalities, i.e., $\mathcal{P} \coloneqq \{y \in \mathbb{R}^p \colon \hat{A}\, y \geqq \hat{a}\}$, with $\hat{A} \in \mathbb{R}^{q \times p}$ and $\hat{a} \in \mathbb{R}^q$. Thereby, q is the finite number of inequalities.*

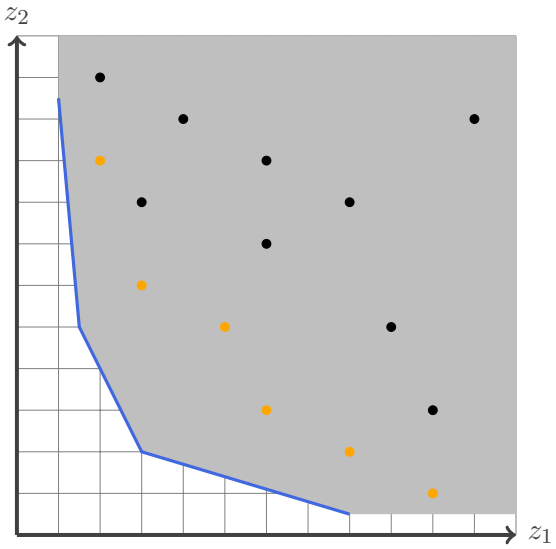
Hence, a polyhedron is the intersection of a finite number of half-spaces and therefore a closed convex set.

Definition 2.16 (Nemhauser and Wolsey, 1988)**.** *A polyhedron $\mathcal{P} \subseteq \mathbb{R}^p$ is called bounded, if there exists a $w > 0$ such that $\mathcal{P} \subseteq \{y \in \mathbb{R}^p \colon -w \leq y_j \leq w,\ j = 1, \dots p\}$. A bounded polyhedron is called a polytope.*

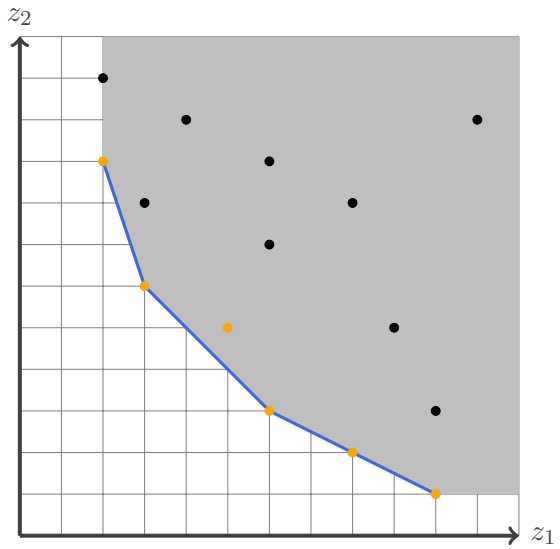
By $\dim(\mathcal{P})$ we denote the *dimension* of a polyhedron $\mathcal{P}$. Let $k \in \mathbb{Z}$ be the number of affinely independent points in $\mathcal{P}$, then $\dim(\mathcal{P}) = k - 1$.



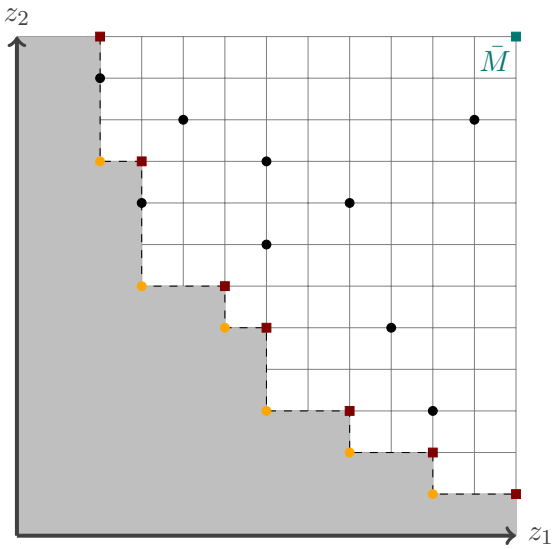
(a) The yellow points represent the set Y_N. The ideal point y^I is a valid lower bound and the nadir point y^N and anti-ideal point y^{AI} are valid upper bounds on them.



(b) The blue line represents the solution of the linear relaxation. It is a valid lower bound set on the set Y_N represented by the yellow points.



(c) The blue line represents the solution of the convex relaxation. It is a valid lower bound set on the set Y_N represented by the yellow points.



(d) The yellow points represent the set Y_N. The red rectangles represent the corresponding set of local upper bounds $\mathcal{D}(Y_N)$. They are a valid upper bound set on the non-dominated points.

Figure 2.2: An Illustration of different bounds and bound sets on the set of non-dominated points Y_N for a bi-objective (MO01LP).

Definition 2.17 (Nemhauser and Wolsey, 1988)**.** *A polyhedron $\mathcal{P} \subseteq \mathbb{R}^p$ is full-dimensional, if* $\dim(\mathcal{P}) = p$.

There are two common ways to describe a polyhedron. Namely, by using *facets* or by using *vertices and rays*.

We start with the description by facets. Given a polyhedron $\mathcal{P} = \{y \in \mathbb{R}^p \colon \hat{A}\, y \geqq \hat{a}\}$, we want to figure out if all of the inequalities are necessary or some can be dropped. An inequality and thus its corresponding half-space $\mathcal{H}^+$ is called *valid* regarding a polyhedron $\mathcal{P}$, if $\mathcal{P} \subseteq \mathcal{H}^+$. Furthermore, a valid inequality $a^\top y \geq a_0$ is called a *face*, if there exist a $\bar{y} \in \mathcal{P}$ with $a^\top \bar{y} = a_0$. A valid inequality which is not a face is called *redundant*. A face $\mathcal{F}$ can also be described as the intersection of a hyperplane $\mathcal{H}$ and the polyhedron $\mathcal{P}$, i.e., $\mathcal{F} \coloneqq \{y \in \mathcal{H} \colon y \in \mathcal{P}\}$, if it is not empty.

Definition 2.18 (Nemhauser and Wolsey, 1988)**.** *A face $\mathcal{F}$ of a polyhedron $\mathcal{P}$ is called facet, if* $\dim(\mathcal{F}) = \dim(\mathcal{P}) - 1$.

For each facet $\mathcal{F}$ of $\mathcal{P}$, one inequality that represents $\mathcal{F}$ is necessary in the description of $\mathcal{P}$. By $\mathcal{P}_\mathcal{H} \coloneqq \{\mathcal{H}_1^+, \ldots, \mathcal{H}_t^+\}$ we denote the *half-space representation* of $\mathcal{P}$, where the half-spaces $\mathcal{H}_1^+, \ldots, \mathcal{H}_t^+$ correspond to the inequalities that describe the t facets of $\mathcal{P}$. The polyhedron $\mathcal{P}$ is then given by

$$\mathcal{P} = \bigcap_{H \in \mathcal{P}_\mathcal{H}} H.$$

Alternatively, instead of describing the polyhedron with the highest-dimensional faces, the polyhedron can be described by using the lowest-dimensional faces.

Definition 2.19 (cf. Nemhauser and Wolsey, 1988)**.** *A face $\mathcal{F}$ of a polyhedron $\mathcal{P}$ is called vertex, if* $\dim(\mathcal{F}) = 0$.

Additionally, we need to define rays. A vector $r \in \mathbb{R}^p \setminus \{0\}$ is called *ray* of $\mathcal{P}$ if for all $y \in \mathcal{P}$ the condition $\{\bar{y} = y + \lambda r,\ \lambda \geq 0\} \subseteq \mathcal{P}$ is fulfilled.

Definition 2.20 (cf. Nemhauser and Wolsey, 1988)**.** *A ray $r \in \mathbb{R}^p$ is an extreme ray, if there do not exist rays $r^1, r^2 \in \mathbb{R}^p$ of $\mathcal{P}$ with $r^1 \neq \gamma r^2$ for any $\gamma \geq 0$, such that $r = \lambda_1 r^1 + \lambda_2 r^2$ with $\lambda_1, \lambda_2 > 0$.*

Every facet $\mathcal{F}$ of a polyhedron $\mathcal{P}$ can be described by using a finite set of vertices $V_\mathcal{F}$ and a finite set of extreme rays $R_\mathcal{F}$. These sets need to satisfy $\mathcal{F} = \operatorname{conv}(V_\mathcal{F}) + \{\sum_{r \in R_\mathcal{F}} \lambda_r r,\ \lambda \geqq 0\}$. Therefore, we can introduce the *vertex-ray representation* of a polyhedron $\mathcal{P}$ denoted by $\mathcal{P}_{VR} = (V_\mathcal{P}, R_\mathcal{P})$, where $V_\mathcal{P}$ is the set of vertices of $\mathcal{P}$ and $R_\mathcal{P}$ is the set of extreme rays

of $\mathcal{P}$. The polyhedron $\mathcal{P}$ is then given by

$$\mathcal{P} = \operatorname{conv}(V_{\mathcal{P}}) + \left\{ \sum_{r \in R_{\mathcal{P}}} \lambda_r r,\ \lambda \geqq 0 \right\}.$$

Obviously, if a polyhedron $\mathcal{P}$ has no extreme rays, i.e., $R_{\mathcal{P}} = \emptyset$, it is an polytope. Many algorithms keep track of the facets as well as the vertices and rays, denoted by $(\mathcal{P}_{\mathcal{H}}, \mathcal{P}_{VR})$. This approach is called *double description method* (see, e.g., Fukuda and Prodon, 1996). Obviously, it is necessary to link both sets. Otherwise it is not clear which vertices and rays correspond to which hyperplanes. The linking can be done by for example using an adjacency matrix. This result is given by the well-known Minkowski-Weyl Theorem.

Theorem 2.21 (Minkowski-Weyl Theorem, cf. Ziegler, 1995)**.** *For $\mathcal{P} \subseteq \mathbb{R}^p$, the following statements are equivalent:*

i) $\mathcal{P}$ is a polyhedron, i.e., there exists some halfspace representation $\mathcal{P}_{\mathcal{H}}$, such that

$$\mathcal{P} = \bigcap_{H \in \mathcal{P}_{\mathcal{H}}} H.$$

ii) There is a vertex-ray representation $\mathcal{P}_{VR}$ such that

$$\mathcal{P} = \operatorname{conv}(V_{\mathcal{P}}) + \left\{ \sum_{r \in R_{\mathcal{P}}} \lambda_r r,\ \lambda \geqq 0 \right\}.$$

3 Solution Methods

In this chapter, we introduce some of the most relevant solution methods for multi-objective integer linear problems. These solution approaches are often categorized in: *objective space methods* and *decision space methods*.

In the first section, we present a selection of well-known *scalarization methods*. Thereby, we clarify whether the generated solutions are guaranteed to be efficient. Moreover, it is described whether all non-dominated points can be found with each specific approach or whether just a subset of Y_N is generated.

In the second section, we restrict ourselves to *multi-objective dynamic programming* and the *multi-objective branch and bound method*, since these are the most widely used decision space methods. Particularly for the branch and bound method, each single component is described in detail and a sufficient overview of the corresponding literature is given.

3.1 Objective Space Methods

The so-called objective space methods scalarize the underlying problem, i.e., it is replaced by one or a series of single-objective problems to determine successively the set of efficient solutions. In the case of multi-objective integer programming, these scalarized problems can be solved with commercial single-objective integer programming solvers like CPLEX[1] or Gurobi[2]. The utilization of these optimized singlecriteria solvers are a major advantage and one of the reasons why these methods are predominant in comparison to the decision space methods in multi-objective optimization.

A large variety of these methods is proposed in the literature but we restrict ourselves to the best known approaches.

3.1.1 ε-constraint Method

The *ε-constraint method* was introduced for two objectives by Haimes et al. (1971) and has been further discussed in Chankong and Haimes (1983). In this method, one of the objective functions $z_k, k \in \{1, \ldots, p\}$ of (MO01LP) is selected as the objective function of the scalarized problem. The remaining $p-1$ objective functions are transformed into

[1] https://www.ibm.com/products/ilog-cplex-optimization-studio
[2] https://www.gurobi.com

constraints that bound the corresponding objective values. Hence, the ε-constraint scalarization can be written in the form:

$$\begin{aligned} \min\ & z_k(x) \\ \text{s.t.}\ & z_j(x) \leq \varepsilon_j \quad \forall j = 1, \ldots, p,\ j \neq k \\ & x \in X. \end{aligned} \tag{ε^k-C}$$

Proposition 3.1 (cf. Ehrgott, 2005)**.** *Let $\bar{x}$ be an optimal solution of (ε^k-C) for some $k \in \{1, \ldots, p\}$, i.e., $\bar{x} \in X_k(\varepsilon)$. Then $\bar{x}$ is weakly efficient.*

Proof. Assume $\bar{x} \notin X_{WE}$. Then there is an $x \in X$ such that $z_j(x) < z_j(\bar{x})$ for all $j = 1, \ldots, p$. In particular, $z_k(x) < z_k(\bar{x})$. Since $z_j(x) < z_j(\bar{x}) \leq \varepsilon_j$ for all $j \neq k$, the solution x is feasible for (ε^k-C). This is a contradiction to $\bar{x}$ being an optimal solution of (ε^k-C). □

Note that if $\bar{x}$ is an unique solution it is efficient (Ehrgott, 2005). Furthermore, it is possible to generate all non-dominated points with this scalarization technique.

Proposition 3.2 (cf. Ehrgott, 2005)**.** *The feasible solution $\bar{x} \in X$ is efficient if and only if there exists an $\bar{\varepsilon} \in \mathbb{R}^p$ such that $\bar{x} \in X_k(\bar{\varepsilon})$ for all $k = 1, \ldots, p$.*

Proof.

First assume $\bar{x} \notin X_k(\bar{\varepsilon})$ for some $k \in \{1, \ldots, p\}$ with $\bar{\varepsilon} = z(\bar{x})$. Then, there must be some $x \in X$ with $z_k(x) < z_k(\bar{x})$ and $z_j(x) \leq z_j(\bar{x}) = \bar{\varepsilon}_j$ for all $j \neq k$, which would imply $\bar{x} \notin X_E$.

On the contrary, suppose $\bar{x} \notin X_E$. Then there is an index $k \in \{1, \ldots, p\}$ and a feasible solution $x \in X$ such that $z_k(x) < z_k(\bar{x})$ and $z_j(x) \leq z_j(\bar{x})$ for all $j \neq k$. Therefore $\bar{x}$ can not be an optimal solution of (ε^k-C) for any $\varepsilon \in \mathbb{R}^p$ for which it is feasible. □

In every iteration of this method, the k-th objective is optimized with updated ε-values to ensure improvement regarding the other objectives. Afterwards, the weakly non-dominated points have to be filtered out. In Laumanns et al. (2006), an extension to three and more objectives is presented. Many approaches based on the ε-constraint method have been published in the last decades. For example Boland et al. (2017) and Kirlik and Sayın (2014) combine the method with reduction of dimension in the tri- respectively multi-dimensional case. There are also approaches that guarantee the efficiency of the generated solutions (see, e.g., Kirlik and Sayın, 2014; Mavrotas, 2009; Özlen and Azizoğlu, 2009). The two common ways to do this is by adding an augmentation term to the objective function (see, e.g., Özlen and Azizoğlu, 2009) or by using a so-called *lexicographic optimization* (see, e.g., Kirlik and Sayın, 2014; Mavrotas, 2009). Other approaches, see for example Bérubé et al. (2009), improve the choice of the values for ε.

For two objectives the choice for ε is relatively straightforward, since it can be derived from the previous iteration of this method. Given a non-dominated point $\bar{y} \in \mathbb{R}^2$. Without loss of generality we choose $k = 1$, i.e., we minimize the first objective. The next (weakly) non-dominated point with respect to z_1 (if it exists) can be found by solving (ε^k-C) with $\varepsilon_2 = \bar{y}_2 - \delta$. Note that in general $\delta > 0$ should be chosen sufficiently small, but since this work only considers integer programming with integral coefficients $\delta = 1$ is a valid choice. For three and more objectives this approach is not straightforward anymore. Since the natural order of the set of non-dominated points is lost in $p \geq 3$ it is more difficult to keep track of the search space which still needs to be examined. We refer to Dächert et al. (2017) for an detailed overview of the difficulties and computation of the search region. Furthermore, it is possible that numerous of ε-constraint scalarizations are infeasible and therefore harder to solve. In Laumanns et al. (2006), an extension of the adaptive choice to the multi-objective case is proposed. An ε-constraint method for two objectives is presented in Algorithm 1.

Algorithm 1: Bi-objective ε-constraint Method

1 **Input:** Initial problem (MO01LP)
2 **Output:** Set of non-dominated points Y_N
3 $x^1 \coloneqq \operatorname{argmin}\{z_1(x) \colon x \in X,\, z_2(x) \leq \varepsilon_2\}$ with $\varepsilon_2 = M$.
4 $\bar{X} \coloneqq \{x^1\}$
5 Set $\varepsilon_2 \coloneqq z_2(x^1) - 1$.
6 **while** (ε^k-C) *is feasible* **do**
7 Determine $\bar{x} \coloneqq \operatorname{argmin}\{z_1(x) \colon x \in X,\, z_2(x) \leq \varepsilon_2\}$.
8 $\bar{X} \coloneqq \bar{X} \cup \{\bar{x}\}$.
9 Set $\varepsilon_2 \coloneqq z_2(\bar{x}) - 1$.
10 $X_E \coloneqq \{x \in \bar{X} \colon z(x) \text{ is non-dominated}\}$
11 $Y_N \coloneqq \{z(x) \in \mathbb{R}^2 \colon x \in X_E\}$

Note that Algorithm 1 can be adjusted by adding a lexicographic reoptimization step between line 7 and line 8 (see, e.g., Mavrotas, 2009) to guarantee efficiency. Therefore, after determining $\bar{x}$ we determine $\hat{x} \coloneqq \operatorname{argmin}\{z_2(x) \colon x \in X,\, z_1(x) = z_1(\bar{x})\}$ and add $\hat{x}$ to $\bar{X}$.

Example 3.3. *In Figure 3.1, an illustration of the procedure of Algorithm 1 is given. In Figure 3.1(a), the set Y of an* (MO01LP) *with $p = 2$ is visualized by the black points. The yellow point $z(\bar{x}) \in Y$ is obtained by solving* (ε^k-C) *with $\varepsilon_2 = M$. In Figure 3.1(b), we see that z_2 is bounded by $\varepsilon_2 = z_2(\bar{x}) - 1$. This is visualized by the gray area, which shows the now infeasible parts of Y. The yellow point is obtained by solving the corresponding ε-constraint scalarization. In Figure 3.1(c) to Figure 3.1(g), this procedure is repeated.*

Then in Figure 3.1(h), the ε-constraint scalarization is infeasible, i.e., the while-loop stops. Since all non-dominated points are among the set of obtained points, we can filter those for dominance. The final set of non-dominated points is visualized in Figure 3.1(i).

3.1.2 Weighted Sum Method

The *weighted sum method* is an objective space method based on the optimization of a weighted sum of the p objective functions using positive weights. It was first formally discussed in Gass and Saaty (1955). The weighted sum scalarization can be formulated as

$$\begin{aligned} \min \ & \sum_{j=1}^{p} \lambda_j \, z_j(x) \\ \text{s.t.} \ & x \in X, \end{aligned} \tag{WS$_\lambda$}$$

with $\lambda \in \mathbb{R}^p_{\geq}$ and $\sum_{j=1}^{p} \lambda_j = 1$. The values $\lambda_j, j = 1, \ldots, p$ model the relative importance among the objectives. Contrary to the ε-constraint scalarization, the set of feasible solutions remains unchanged. This assures that the constraint structure does not change, which is particularly important for combinatorial optimization problems. Geoffrion (1968) showed that an optimal solution of (WS$_\lambda$) is guaranteed to be efficient, respectively weakly efficient, depending on the choice of λ.

In the following, we use the sets $\Lambda_0 \coloneqq \{\lambda \in \mathbb{R}^p \colon \lambda \geq 0, \sum_{j=1}^{p} \lambda_j = 1\}$ and $\Lambda \coloneqq \{\lambda \in \mathbb{R}^p \colon \lambda > 0, \sum_{j=1}^{p} \lambda_j = 1\}$ to denote the respective weight-spaces and thus ease the notation.

Proposition 3.4 (Ehrgott, 2005)**.** *Let $\bar{x}$ be an optimal solution of* (WS$_\lambda$) *with $\lambda \in \mathbb{R}^p_{\geq}$. Then the following statements hold.*

i) If $\lambda \in \Lambda_0$, then $\bar{x}$ is weakly efficient.

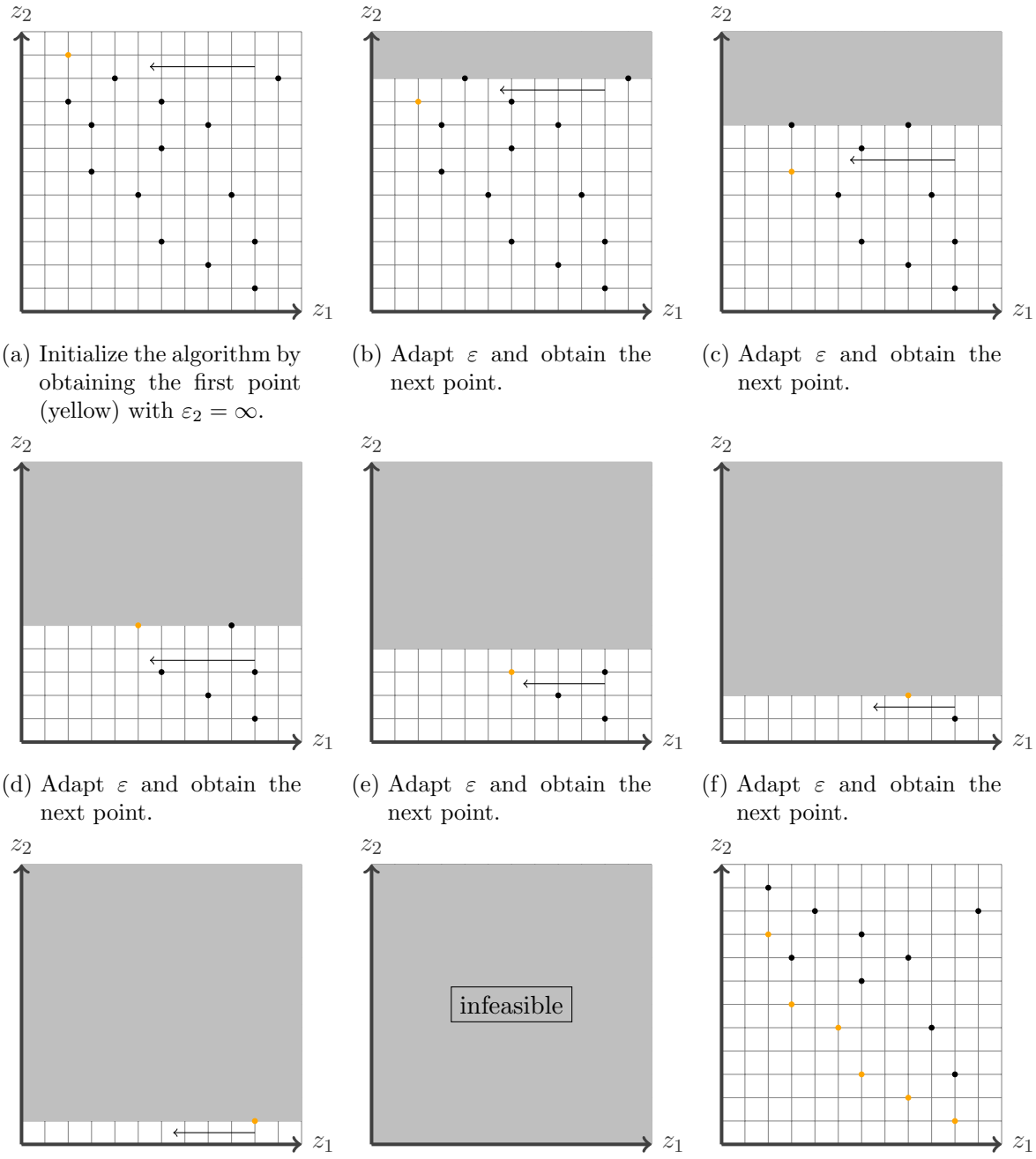
ii) If $\lambda \in \Lambda$, then $\bar{x}$ is efficient.

iii) If $\lambda \in \Lambda_0$ and $\bar{x}$ is a unique optimal solution of (WS$_\lambda$)*, then $\bar{x}$ is efficient and $z(\bar{x}) \in Y_{SN1}$.*

Proof.

i) Assume $\bar{x} \notin X_{WE}$. Then there is an $x \in X$ such that $z(x) < z(\bar{x})$. Hence, $\lambda^\top z(x) < \lambda^\top z(\bar{x})$ also holds for all $\lambda \in \Lambda_0$. This is a contradiction to $\bar{x}$ being an optimal solution of (WS$_\lambda$).

ii) Assume $\bar{x} \notin X_E$. Then there is an $x \in X$ such that $z_j(x) \leq z_j(\bar{x})$ for all $j = 1, \ldots, p$ and $z_k(x) < z_k(\bar{x})$ for at least one $k \in \{1, \ldots, p\}$. Hence, $\lambda^\top z(x) \leq$



(a) Initialize the algorithm by obtaining the first point (yellow) with $\varepsilon_2 = \infty$.

(b) Adapt ε and obtain the next point.

(c) Adapt ε and obtain the next point.

(d) Adapt ε and obtain the next point.

(e) Adapt ε and obtain the next point.

(f) Adapt ε and obtain the next point.

(g) Adapt ε and obtain the next point.

(h) The ε-constraint scalarization is infeasible. The while-loop stops.

(i) Filter the obtained points by dominance.

Figure 3.1: A bi-objective example to illustrate the procedure of a the ε-constraint method given in Algorithm 1. The obtained point of each iteration is depicted in yellow. The gray area represents the bounding of the second objective z_2.

$\lambda^\top z(\bar{x})$ holds for all $\lambda \in \Lambda$. This is a contradiction to $\bar{x}$ being an optimal solution of (WS_λ).

iii) Assume $\bar{x} \notin X_E$. Then, there is an $x \in X$ such that $z_j(x) \leq z_j(\bar{x})$ for all $j = 1, \ldots, p$ and $z_k(x) < z_k(\bar{x})$ for at least one $k \in \{1, \ldots, p\}$. Consequently, $\lambda^\top z(x) \leq \lambda^\top z(\bar{x})$ holds for all $\lambda \in \Lambda_0$. This is contradicting to $\bar{x}$ being the unique optimal solution of (WS_λ).

□

Since we are only interested in efficient solutions, we can restrict the considered weight space for linear multi-objective problems to Λ. Note that this is not possible for non-linear continuous optimization problems as there might be non-dominated points with unbounded trade-off. Feasible outcome vectors $y \in Y$ whose corresponding efficient solution $x \in X_E$ with $z(x) = y$ can be obtained as optimal solution of (WS_λ) fo some weighting vector $\lambda \in \Lambda$ are denoted as supported non-dominated points (Ehrgott, 2005). Since in general not all non-dominated points are supported, i.e., $Y_N \neq Y_{SN}$, not all non-dominated points can be obtained by solving weighted sum scalarizations. Regardless of the chosen weights it is not possible to obtain unsupported points $Y_{UN} = Y_N \backslash Y_{SN}$ by solving (WS_λ). In addition, although there exist weights such that $\bar{x}$ with $z(\bar{x}) \in Y_{SN2}$ is optimal for the weighted sum scalarization, it is not guaranteed that a single-objective solver finds this solution. In those cases, there is more than one feasible solution with identical optimal objective value of (WS_λ) but a solver returns only one these solutions.

Despite this limitation there are several weighted sum approaches proposed in the literature. Aneja and Nair (1979) developed a bi-objective weighted sum method with a *dichotomic search scheme*. This scheme determines weights adaptively based on the convex hull of already computed non-dominated points and guarantees to find all points in Y_{SN1}. Extensions of this approach to the multi-objective case with $p \geq 3$ are proposed in Przybylski et al. (2010a) and Przybylski et al. (2019). Although the dichotomic approach is the gold standard for two objectives, its extension to more objectives is not straightforward, since the computation of the convex hull is considerably more difficult and numerically instable in higher dimensions. Moreover, the weight determined in the dichotomic search, might have negative components. Thus, solving the corresponding weighted sum scalarization would, in general, result in a dominated point. The authors overcome this problem by using different ways to initialize the algorithm by computing the boundary of the non-dominated set and determining the remaining set of non-dominated points in a specific order. Further approaches for the multi-objective case can be found for example in Özpeynirci and Köksalan (2010) and Bökler and Mutzel (2015). A weighted

sum method for two objectives (based on Aneja and Nair, 1979) is given in the recursive Algorithm 2.

Algorithm 2: Bi-objective Weighted Sum Method (dichotomic search)

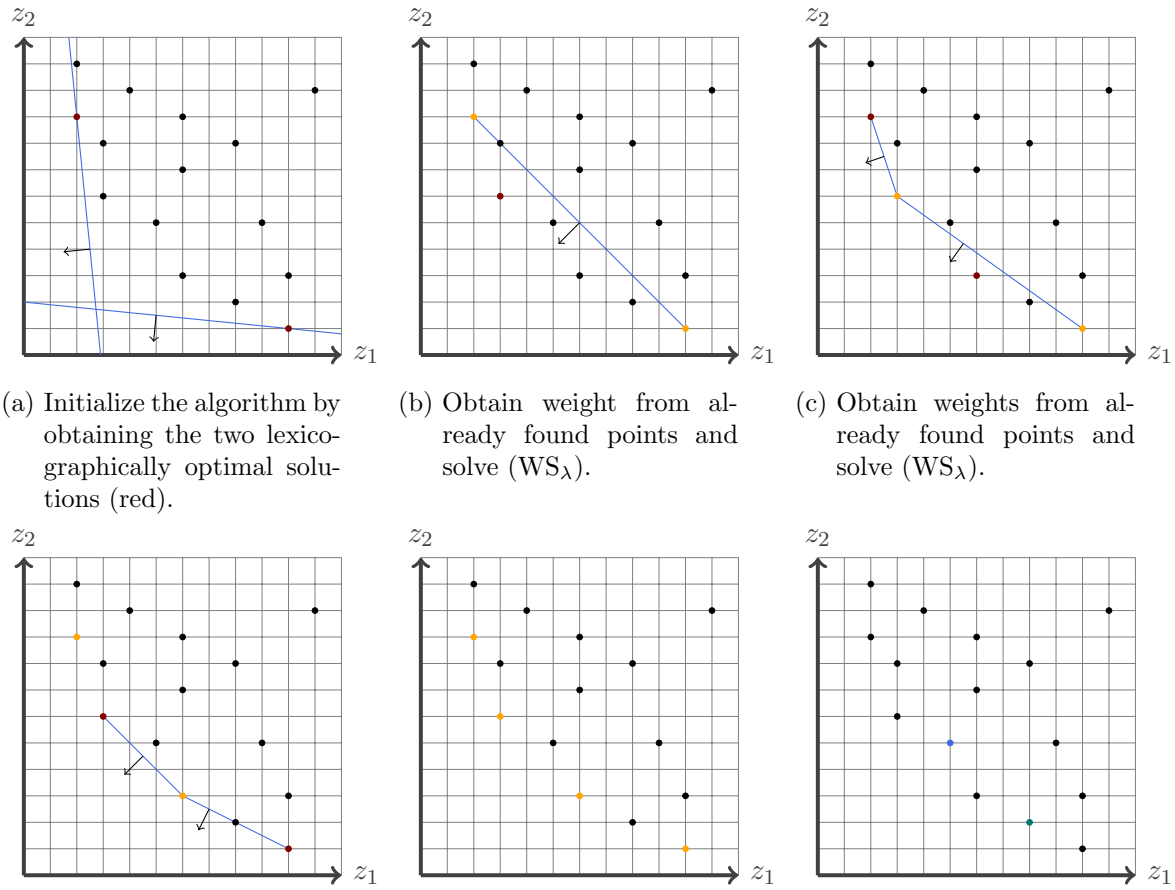
Input: Initial problem (MO01LP)
Output: Subset of supported non-dominated points $Y_{WS} \subseteq Y_{SN}$
$x^i \coloneqq \operatorname{argmin}\{(\lambda^i)^\top z(x),\, x \in X\}$ for $i = 1, 2$ with $\lambda^1 \coloneqq (1-\delta, \delta)$ and $\lambda^2 \coloneqq (\delta, 1-\delta)$, where $\epsilon > 0$ is sufficiently small.
$\bar{X} \coloneqq \{x^1, x^2\}$
$\bar{X} \coloneqq$ `solveRecursion`$(x^1, x^2, \bar{X})$
$Y_{WS} \coloneqq \{z(x) \in \mathbb{R}^2 \colon x \in \bar{X}\}$

Procedure: `solveRecursion`

Input: $x^1, x^2 \in X_E, \bar{X} \subseteq X_E$
Output: $\bar{X} \subseteq X_E$
$\lambda_1 \coloneqq z_2(x^1) - z_2(x^2), \lambda_2 \coloneqq z_1(x^2) - z_1(x^1)$
$\bar{x} \coloneqq \operatorname{argmin}\{\lambda^\top z(x),\, x \in X\}$.
$\bar{X} \coloneqq \bar{X} \cup \bar{x}$.
if $\lambda_1 z_1(\bar{x}) + \lambda_2 z_2(\bar{x}) < \lambda_1 z_1(x^1) + \lambda_2 z_2(x^1)$ **then**
    $\bar{X} \coloneqq$ `solveRecursion`$(x^1, \bar{x}, \bar{X})$
    $\bar{X} \coloneqq$ `solveRecursion`$(\bar{x}, x^2, \bar{X})$

Example 3.5. *In Figure 3.2, an illustration of Algorithm 2 can be found. The set of feasible points Y of a* (MO01LP) *with two objectives is visualized by the black points. In Figure 3.2(a), the two lexicographically optimal points are computed (red) by solving* (WS_λ) *with weights $\lambda^1 = (1-\delta, \delta)^\top$ and $\lambda^2 = (\delta, 1-\delta)^\top$ where $\delta > 0$ is sufficient small. These two points define a hyperplane in Figure 3.2(b) and its corresponding normal (with strictly positive components) is chosen as weight. Solving this weighted sum scalarization yields a new non-dominated point (red). Note that non-dominated points which were found in prior iterations are depicted in yellow. In Figure 3.1(c), two new hyperplanes are obtained and two weighted sum scalarizations are solved. Again, the red points depict the corresponding objective vector of the obtained solutions. Note that one of the solved scalarizations does not find a new non-dominated point. Therefore, this area can be neglected in following iterations. In Figure 3.2(d), again, we obtain two new hyperplanes and therewith two new weighted sum scalarizations that need to be solved. Since no new non-dominated points are found, the algorithm stops. Note that it would be possible to find a new non-dominated point in this iteration. But since this point is in Y_{SN2}, i.e., it is a supported non-dominated*

point but not an supported non-dominated extreme point, it cannot be guaranteed that this point is found by the single-objective solver. In Figure 3.2(e), the subset of Y_{SN} that is obtained by solving this example is shown. Figure 3.2(e) visualizes all other non-dominated points that were not found with this method. The blue point is unsupported and therefore it was clear in advance, that it is not possible to obtain this point with the weighted sum method. The green point is supported but since it is not extreme there is no guarantee that it is found.



(a) Initialize the algorithm by obtaining the two lexicographically optimal solutions (red).

(b) Obtain weight from already found points and solve (WS_λ).

(c) Obtain weights from already found points and solve (WS_λ).

(d) Obtain weights from already found points and solve (WS_λ).

(e) Subset of Y_{SN} obtained by using the weighted sum method.

(f) Non-dominated points that were not found by using the weighted sum method.

Figure 3.2: A bi-objective example to illustrate the procedure of a the weighted sum method given in Algorithm 2. The obtained points of each iteration is depicted in red. Points that have been found in previous iterations are illustrated in yellow.

3.1.3 Augmented Weighted Tchebycheff Method

The *augmented weighted Tchebycheff method* belongs to the class of so-called *reference point methods*. The objective function of the corresponding scalarizations is given by minimizing the distance to a predefined reference point $s \in \mathbb{R}^p$. Often norm induced distance measures are applied. A corresponding reference point scalarization using a norm $\|.\|$ can be written as

$$\begin{aligned} \min \ & \|z(x) - s\| \\ \text{s.t.} \ & x \in X. \end{aligned} \tag{$\mathrm{RP}_{\|.\|}$}$$

The properties of the scalarization depend on the choice of the considered norm.

Definition 3.6. *Let $\|.\|\colon \mathbb{R}^p \to \mathbb{R}_{\geqq}$ be a norm in $\mathbb{R}^p$, i.e.,*

i) $\|y\| = 0 \Leftrightarrow y = 0$

ii) $\|\alpha y\| = |\alpha| \cdot \|y\|, \quad \forall\, y \in \mathbb{R}^p,\ \forall\, \alpha \in \mathbb{R}$

iii) $\|y^1 + y^2\| \leq \|y^1\| + \|y^2\|, \quad \forall\, y^1, y^2 \in \mathbb{R}^p.$

Then, the norm $\|.\|$ is called

1. *monotone, if*

   i) $\|y^1\| \leq \|y^2\| \quad \forall\, y^1, y^2 \in \mathbb{R}^p\colon \quad |y^1_j| \leq |y^2_j| \quad \forall\, j = 1, \ldots, p$ *and*

   ii) $\|y^1\| < \|y^2\| \quad \forall\, y^1, y^2 \in \mathbb{R}^p\colon \quad |y^1_j| < |y^2_j| \quad \forall\, j = 1, \ldots, p.$

2. *strictly monotone, if*

   i) $\|y^1\| < \|y^2\| \quad \forall\, y^1, y^2 \in \mathbb{R}^p\colon \quad |y^1_j| \leq |y^2_j| \quad \forall\, j = 1, \ldots, p$ *and* $\exists\, k \in \{1, \ldots, p\}\colon |y^1_k| < |y^2_k|$.

Proposition 3.7 (cf. Miettinen, 1998)**.** *Let $\bar{x}$ be an optimal solution of* ($\mathrm{RP}_{\|.\|}$).

i) If $\|.\|$ is monotone then $\bar{x}$ is weakly efficient.

ii) If $\|.\|$ is monotone and $\bar{x}$ is an unique optimal solution then $\bar{x}$ is efficient.

iii) If $\|.\|$ is strictly monotone then $\bar{x}$ is efficient.

Proof.

i) Suppose $\bar{x} \notin X_{WE}$. Then, there is an $x \in X$ with $z(x) < z(\bar{x})$ and therefore $z_j(x) - s_j < z_j(\bar{x}) - s_j$ holds for all $j \in \{1, \ldots, p\}$. Hence, $\|z(x) - s\| < \|z(\bar{x}) - s\|$ also holds due to the monotonicity of $\|.\|$ which is contradicting to the optimality of $\bar{x}$.

ii) Suppose $\bar{x} \notin X_E$. Then, there is an $x \in X$ such that $z_j(x) \leq z_j(\bar{x})$ for all $j = 1, \ldots, p$ and $z_k(x) < z_k(\bar{x})$ for at least one $k \in \{1, \ldots, p\}$. Hence, $z_j(x) - s_j \leq z_j(\bar{x}) - s_j$ holds for all $j \in \{1, \ldots, p\}$, which implies $\|z(x) - s\| \leq \|z(\bar{x}) - s\|$ since $\|.\|$ is monotone. This is contradicting the unique optimality of $\bar{x}$.

iii) Suppose $\bar{x} \notin X_E$. Then, there is an $x \in X$ such that $z_j(x) \leq z_j(\bar{x})$ for all $j = 1, \ldots, p$ and $z_k(x) < z_k(\bar{x})$ for at least one $k \in \{1, \ldots, p\}$. Hence, $z_j(x) - s_j \leq z_j(\bar{x}) - s_j$ holds for all $j \in \{1, \ldots, p\}$. Since $\|.\|$ is a strictly monotone norm, $\|z(x) - s\| < \|z(\bar{x}) - s\|$ holds which is a contradiction to the optimality of $\bar{x}$.

□

The *weighted Tchebycheff method* is an objective space method based on the monotone *weighted Tchebycheff maximum norm* and has been introduced by Bowman (1976). First, we introduce the weighted Tchebycheff norm. Let $w_j > 0,\ j = 1, \ldots, p$ be positive weights with $\sum_{j=1}^{p} w_j = 1$. Then the weighted Tchebycheff norm of a vector $y \in \mathbb{R}^p$ is defined by

$$\|y\|_\infty^w := \max_{j=1,\ldots,p} \left\{ w_j \cdot |y_j| \right\}. \tag{3.1}$$

So, the weighted Tchebycheff scalarization of a multi-objective problem (MO01LP) with respect to a given reference point $s \in \mathbb{R}^p$ can be written as:

$$\begin{aligned} \min\ & \|z(x) - s\|_\infty^w \\ \text{s.t.}\ & x \in X. \end{aligned} \tag{WT^w}$$

The reference point has to be chosen such that $s < z(x)$ for all $x \in X$. Otherwise, it is not possible to determine all efficient solutions of the underlying multi-objective problem (Miettinen, 1998). A point $s \in \mathbb{R}^p$ with these properties is also called an *utopian point* as it strictly dominates the ideal point. The absolute values in (3.1) can be neglected, since $s_j < z_j(x),\ j = 1, \ldots, p$ for all $x \in X$.

According to Proposition 3.7, the optimal solutions of (WT^w) are in general only weakly efficient. To overcome this drawback, several extensions of the weighted Tchebycheff method have been developed which assure that the obtained solutions are efficient for (MO01LP) (see, e.g., Kaliszewski, 1987; Steuer and Choo, 1983). One of them is the so-called *augmented weighted Tchebycheff method* (Steuer and Choo, 1983). This method relies on a slightly different scalarization where an augmentation term is added to the weighted Tchebycheff norm which makes it a strictly monotone norm. The *augmented weighted Tchebycheff norm* is defined as

$$\|z\|_\tau^w := \|z\|_\infty^w + \tau \|z\|_1, \tag{3.2}$$

where $\|z\|_1 = |z_1| + \ldots + |z_p|$ denotes the L_1-norm, $w_j > 0$, $j = 1, \ldots, p$, $\sum_{j=1}^{p} w_j = 1$ and $\tau > 0$. The augmented weighted Tchebycheff scalarization can now be written as

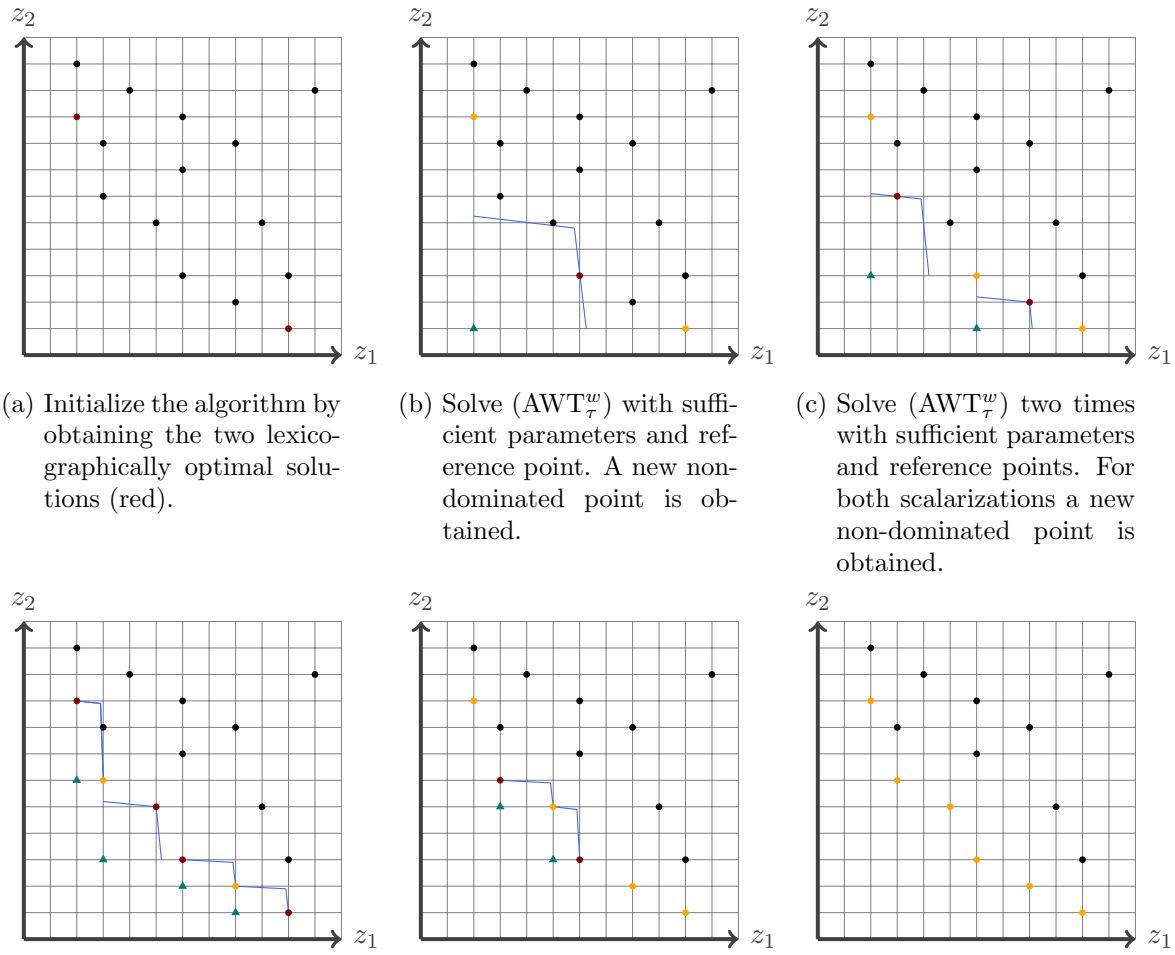
$$\begin{aligned} \min \ & \|z(x) - s\|_\tau^w \\ \text{s.t.} \ & x \in X. \end{aligned} \tag{AWT$_\tau^w$}$$

According to Proposition 3.7, the obtained optimal solutions of (AWT$_\tau^w$) are efficient and it is shown by Steuer and Choo (1983) that for every efficient solution $\bar{x}$ there are corresponding weights w and τ such that $\bar{x}$ is an optimal solution of (AWT$_\tau^w$). But an appropriate choice of the parameter τ is difficult in general. On the one hand, too small values of τ can lead to numerical difficulties. On the other hand, non-supported efficient solutions might be suboptimal for (AWT$_\tau^w$) if the value of τ is too large. However, for bi-objective integer programming Dächert et al. (2012) propose an adaptive method to determine an optimal value of τ. An augmented weighted Tchebycheff method for two objectives is given in Algorithm 4. In this algorithm, the lexmin operator is used to find the solution corresponding to the lexicographically optimal point $z^\pi(x)$ for a given permutation π. For the computation of w and τ we refer to Dächert et al. (2012).

Algorithm 4: Bi-objective Augmented Weighted Tchebycheff Method

Input: Initial problem (MO01LP)
Output: Set non-dominated points Y_N

$x^i \coloneqq \text{lexmin}\{z^{\pi^i}(x) \colon x \in X\}$ for $i = 1, 2$ with $\pi^1 \coloneqq (1, 2)$, $\pi^2 \coloneqq (2, 1)$.
if $z(x^1) = z(x^2)$ **then**
  $\bar{X} \coloneqq \{x^1\}$
else
  $\bar{X} \coloneqq \{x^1, x^2\}$
  $ND \coloneqq \{z(x^1), z(x^2)\}$
  while $\exists$ *adjacent pair of points* z^i, z^{i+1} *in* ND *not yet investigated* **do**
    Compute parameters w and τ and optionally the new reference point s w.r.t. z^i and z^{i+1}.
    $\bar{x} \coloneqq \text{argmin}\{\|z(x) - s\|_\tau^w \colon x \in X\}$.
    if $z(\bar{x}) = z^i$ *or* $z(\bar{x}) = z^{i+1}$ **then**
      Label pair of z^i, z^{i+1} as investigated.
    else
      $\bar{X} \coloneqq \bar{X} \cup \bar{x}$
      $ND \coloneqq ND \cup z(\bar{x})$
$X_E \coloneqq \bar{X}$.
$Y_N \coloneqq ND$



(a) Initialize the algorithm by obtaining the two lexicographically optimal solutions (red).

(b) Solve (AWT_τ^w) with sufficient parameters and reference point. A new non-dominated point is obtained.

(c) Solve (AWT_τ^w) two times with sufficient parameters and reference points. For both scalarizations a new non-dominated point is obtained.

(d) (AWT_τ^w) is solved four times with sufficient parameters and reference points. Only one of the scalarizations yields a new non-dominated point.

(e) (AWT_τ^w) is solved two times with sufficient parameters and reference points. No new point is found.

(f) Set of non-dominated points Y_N.

Figure 3.3: A bi-objective example to illustrate the procedure of a the augmented weighted Tchebycheff method given in Algorithm 4. The obtained points of each iteration are depicted in red. Points that have been found in previous iterations are illustrated in yellow.

Example 3.8. *In Figure 3.3, an illustration of solving a bi-objective* (MO01LP) *with Algorithm 4 is given. The set of feasible points Y is given by the black points. In Figure 3.3(a), the algorithm is initialized by obtaining the two lexicoraphically optimal solutions (red). Those currently adjacent points define the reference point (green) in Figure 3.3(b). Solving* (AWT_τ^w) *with sufficient parameters yields a new non-dominated point (red). In Figure 3.3(c), we can see that* (AWT_τ^w) *is solved two times with the corresponding computed reference points and parameters. In each scalarization a new non-dominated point is found (red). This procedure is repeated in Figure 3.3(d) where four scalarizations with corresponding weights are solved. Since three of these scalariaztions yield points that have been found already, only one new non-dominated point is found. In Figure 3.3(e), no new point is obtained and therefore the algorithm comes to an end. Figure 3.3(f) visualizes the set of non-dominated points Y_N (yellow).*

3.1.4 Search Region Splitting Methods

Contrary to the ε-constraint method, where the restriction of the objective space is increased step by step, *search region splitting methods* subdivide the objective space into different *search zones* that can be explored independently. One of the first search region splitting methods for two objectives is proposed in Chalmet et al. (1986) where the so-called *hybrid approach* is used. The authors combine the weighted sum scalarization with the ε-constraint scalarization. The resulting hybrid scalarization can be formulated as

$$\begin{aligned} \min \ & \sum_{j=1}^{p} \lambda_j z_j(x) \\ \text{s.t.} \ & z_j(x) \leq \varepsilon_j \quad \forall j = 1, \ldots, p \\ & x \in X, \end{aligned} \tag{$\mathrm{HS}_\lambda^\varepsilon$}$$

with $\lambda \in \Lambda_0$ and $\varepsilon \in \mathbb{R}^p$. Since in this scalarization all p objective functions are bounded by ε the objective space is divided into regions, which can be explored independently. Due to the weighted sum objective with $\lambda \in \Lambda_0$ every optimal solution $\bar{x}$ of ($\mathrm{HS}_\lambda^\varepsilon$) is weakly efficient and for every efficient solution $\bar{x}$ there exist suitable λ and $\varepsilon = z(\bar{x})$ such that $\bar{x}$ is an optimal solution of ($\mathrm{HS}_\lambda^\varepsilon$) (Chalmet et al., 1986). Contrary to the weighted sum method, this approach generates all efficient solutions and not only the efficient solutions $\bar{x}$ with $z(\bar{x}) \in Y_{SN}$.

There are numerous approaches of bi-objective search region splitting methods (see, e.g., Hamacher et al., 2007; Leitner et al., 2016). Boland et al. (2015) propose improvements on the search region method of Chalmet et al. (1986). The search region is split up into rectangles that can be explored individually. Each rectangle is then

split in half and those new rectangles are explored individually as well. Furthermore, instead of solving a weighted sum objective, the authors propose a lexicographic optimization in each rectangle. Given two points y^1, y^2 with $y_1^1 < y_1^2$, the corresponding rectangular spanned by those points is defined as $R(y^1, y^2)$. The so-called *corner point* is denoted by $R^C(y^1, y^2) := (y_2^1, y_1^2)$. Note that corner points correspond to local upper bounds or local Nadir points of this rectangle. When a new rectangle is obtained it is split half into the lower part $R_L(y^1, y^2) := R((y_1^1, \frac{1}{2}(y_2^1 + y_2^2)), y^2)$ and the upper part $R_U(y^1, y^2) := (y^1, (y_1^2, \frac{1}{2}(y_2^1 + y_2^2)))$. During the algorithm problems of the following form are solved:

$$\begin{aligned} \text{lexmin } & z^\pi(x) \\ \text{s.t. } & z_j(x) \leq \varepsilon_j \quad \forall j = 1, \dots, p \\ & x \in X. \end{aligned} \tag{LEX-P$_\pi^\varepsilon$}$$

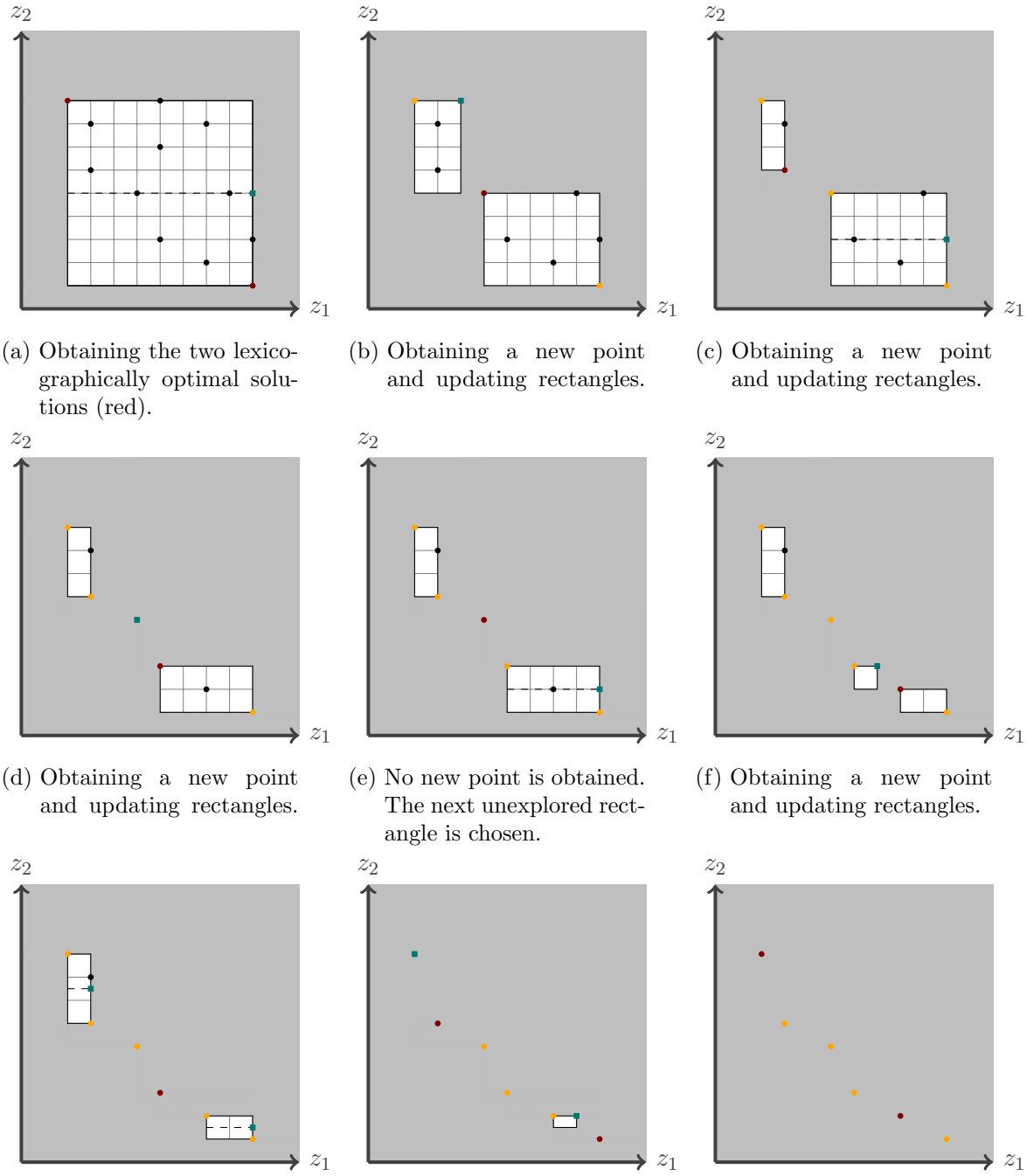
The vector ε is given by the corner point of the rectangle to explore. Again, every optimal solution $\bar{x}$ of (LEX-P$_\pi^\varepsilon$) is efficient and for every efficient solution $\bar{x}$ there exist suitable π and ε such that $\bar{x}$ is the optimal solution of (LEX-P$_\pi^\varepsilon$) (Boland et al., 2015). A bi-objective search region splitting method based on Boland et al. (2015) is given in Algorithm 5.

Algorithm 5: Bi-objective Search Region Splitting Method

Input: Initial problem (MO01LP)
Output: Set of non-dominated points Y_N

$x^i := \text{lexmin}\{z^{\pi^i}(x) \colon x \in X\}$ for $i = 1, 2$ with $\pi^1 := (1, 2)$, $\pi^2 := (2, 1)$.
$\bar{X} := \{x^1, x^2\}$
$L := \{R(z(x^1), z(x^2))\}$
while $L \neq \emptyset$ **do**
    Select and remove a rectangle $R(y^1, y^2)$ from L.
    $\bar{x} := \text{lexmin}\{z^\pi(x) \colon x \in X,\ z(x) \leq \varepsilon\}$ with $\pi := (1, 2)$ and $\varepsilon := R_L^C$.
    if $z(\bar{x}) \neq y^2$ **then**
        $\bar{X} := \bar{X} \cup \{\bar{x}\}$
        $L := L \cup \{R(z(\bar{x}), y^2)\}$
    $\hat{x} := \text{lexmin}\{z^\pi(x) \colon x \in X,\ z(x) \leq \varepsilon\}$ with $\pi := (2, 1)$ and $\varepsilon := R^C(z(\bar{x}) - 1, \frac{1}{2}(y_2^1 + y_2^2))$.
    if $z(\hat{x}) \neq y^1$ **then**
        $\bar{X} := \bar{X} \cup \{\hat{x}\}$
        $L := L \cup \{R(y^1, z(\hat{x}))\}$
$X_E := \bar{X}$.
$Y_N := \{z(x) \colon x \in X_E\}$.

By considering more than two objectives, additional difficulties occur. One of them is the definition, respectively computation, of the corner points. Like already mentioned in



(a) Obtaining the two lexicographically optimal solutions (red).

(b) Obtaining a new point and updating rectangles.

(c) Obtaining a new point and updating rectangles.

(d) Obtaining a new point and updating rectangles.

(e) No new point is obtained. The next unexplored rectangle is chosen.

(f) Obtaining a new point and updating rectangles.

(g) Obtaining a new point. Two rectangles are explored simultaniously.

(h) No new point is obtained. Two rectangles are explored simultaniously.

(i) Set of non-dominated points Y_N.

Figure 3.4: A bi-objective example to illustrate the procedure of a search region splitting method given in Algorithm 5. The obtained points of each iteration are depicted in red. Points that have been found in previous iterations are illustrated in yellow. The corner points are depicted in green.

Section 2.3 these corner points are also referred to as local Nadir points or local upper bounds. Additionally, the subdivison of boxes after finding a new non-dominated point is more difficult. If a box is subdivided into smaller boxes it can occur that for corner points u_1, u_2 it holds $C(u_1) \subset C(u_2)$, making the search zone defined by u_1 redundant. To avoid that, it is proposed to only consider a minimal set of these local upper bounds. Algorithms for the tri-objective and multi-objective case are, e.g., proposed in Dächert and Klamroth (2014) respectively Klamroth et al. (2015). In Dächert et al. (2017), an more efficient way to compute the corner points in higher dimensions is proposed, which is based on a neighborhood structure on the local upper bounds.

Example 3.9. *In Figure 3.4, an example of solving a bi-objective* (MO01LP) *with Algorithm 5 is given. The set of feasible outcome vectors is given by the black points. In Figure 3.4(a), the two lexicographically optimal points are obtained and the corresponding rectangle is spanned. Since there are no non-dominated points outside of this rectangle, these parts can be ignored. For visualization, the parts that can be neglected are colored in gray. The rectangle is split in half by the dashed line and the bottom rectangle is explored first by solving* (LEX-P$_\pi^\varepsilon$) *with suitable* π *and* ε*. The corresponding corner point of the rectangle to explore is depicted by a green square. In Figure 3.4(b), a new non-dominated point is found (red). The rectangles are updated and the upper part of the initial rectangle is explored in the next step. A new non-dominated point (red) is found in Figure 3.4(c). After updating the rectangles, the first iteration is completed and a new rectangle hast to be chosen. In Figure 3.4(c), the bottom rectangle is chosen and it is split in half. Afterwards, the bottom part of the split rectangle is explored with a corresponding corner point (green). A new non-dominated point (red) is found and both parts of the considered splitted rectangle can be updated. Afterwards, the remaining upper rectangle is explored, where the corresponding corner point (green) can be found in Figure 3.4(d). Since the corner point is the only feasible point in the considered rectangle, it is found in Figure 3.4(e). In general, if solving* (LEX-P$_\pi^\varepsilon$) *yields an outcome vector that is already known, the corresponding rectangle can be dismissed, since there are no further non-dominated points inside of it. After updating the rectangles, the second iteration is completed. The same procedure is repeated in Figure 3.4(f) and Figure 3.4(g). For the reason of space, the last two iterations of the algorithm are executed simultaneously in Figure 3.4(g) – Figure 3.4(i), by following the the same routine as in the previous iterations. Since there are no unexplored rectangles left in Figure 3.4(i), the algorithm stops. The set of non-dominated points is visualized in red and yellow. Thereby, the red points are the non-dominated points obtained by solving* (LEX-P$_\pi^\varepsilon$) *in the last iterations.*

Note that Algorithm 5 can be improved by utilizing the integrality of (MO01LP). During

the algorithm it is not necessary to explore the lower part of a splitted rectangle with a width smaller or equal one. Analogously it is not necessary to explore the upper rectangle with a height lower or equal one. Therefore, the algorithm can be improved by reducing the number of solved integer scalarizations.

3.1.5 Two-phase Methods

The so-called *two-phase method* was first introduced by Ulungu and Teghem (1995) for bi-objective optimization problems. As the name suggests, this method runs in two steps. In the first phase, the supported non-dominated extreme points are computed. This can be done by, e.g., using a dichotmic scheme solving weighted sum scalarizations, like it is presented in Section 3.1.2. In the second phase, the remaining non-dominated points are generated by searching in triangles (in the bi-objective case) defined by two neighboring supported non-dominated extreme points. There are different approaches proposed in literature to obtain the remaining non-dominated points in the second phase. Some of them are problem specific and can thus not be applied to general integer problems (see, e.g., Przybylski et al., 2008, 2010b; Tuyttens et al., 2000; Ulungu and Teghem, 1995). Clímaco and Pascoal (2016) propose to use the weighted Tchebycheff scalarization, which has been described in Section 3.1.3, in the second phase. Another way to generate the remaining non-dominated points is to use decision space algorithms (see, e.g., Visée et al., 1998). A detailed description of decision space algorithms, especially the branch and bound method, is presented in Section 3.2.

For a long time, the two-phase method has been restricted to the bi-objective case until Przybylski et al. (2010b) have proposed a two-phase method for assignment problems with more than two objectives.

3.2 Decision Space Methods

Besides their numerous advantages, all objective space methods share a crucial shortcoming when considering MOILPs. Since the scalarized problems are solved by a single-objective solver like CPLEX or Gurobi, most of the information, besides the optimal solution, is lost. So, in each iteration an integer program is solved from scratch, which might lead to a high number of redundant iterations, since information from previous iterations (like bounds, cuts or valid inequalities) are often not transferred to the following iterations. Even though in some objective space methods starting solutions can be transferred from previous iterations, a large number of very similar problems has to be solved. Furthermore, most of these methods require to solve infeasible scalarized integer problems. This might cause high computation times, depending on the problem class.

To overcome this shortcomings, decision space methods have been increasingly investigated in the recent years. In this section, we focus on *multi-objective dynamic programming* and *multi-objective branch and bound*, the two most common decision space approaches.

3.2.1 Multi-objective Dynamic Programming

The dynamic programming approach is a technique to solve MO01LPs that satisfy the *Bellman's Principle of Optimality* (Bellman, 1957). This principle states that optimal solutions of the underlying problem can be derived from optimal solutions of corresponding subproblems. The most prominent problems satisfying Bellman's principle are knapsack and shortest path problems. Dynamic programming relies on a recursive approach. A detailed introduction into single-objective dynamic programming is given in, for example, Bellman (1957), Nemhauser (1966) or Dreyfus (1977).

To obtain the solution of an underlying problem, a dynamic programming algorithm is divided into Γ different stages, with at most Ξ states each. We describe the recursive strategy with an example of a knapsack problem. Given the following problem

$$z^{dp} \coloneqq \max \left\{ \sum_{i=1}^{n} c_i x_i \colon \sum_{i=1}^{n} a_i x_i \leq b_1,\, x \in \{0,1\}^n \right\}, \tag{3.3}$$

with $c, a \in \mathbb{Z}^n_{\geqq}$ and $b \in \mathbb{Z}_{\geq}$, we can define the number number of stages Γ and states Ξ. Every stage corresponds to the decision of fixing a variable to 0 or 1, i.e., there are $\Gamma = n$ stages. The number of states corresponds to the number of feasible left-hand side values of the constraint, i.e., $\Xi = b_1 + 1$. For every stage $\gamma = 1, \ldots, n$ we define $N_\gamma \coloneqq \{1, \ldots, \gamma\}$. Then, we can define the subproblem corresponding to state $\xi = 0, \ldots, b_1$ in stage γ by

$$z_\gamma(\xi) \coloneqq \max \left\{ \sum_{i \in N_\gamma} c_i x_i \colon \sum_{i \in N_\gamma} a_i x_i \leq \xi,\, x \in \{0,1\}^\gamma \right\}. \tag{3.4}$$

Obviously, $z_n(b_1) = z^{dp}$ holds. $z_n(b_1)$ is calculated from z_{n-1} which is respectively calculated from z_{n-2} and so on. The recursion is initialized by

$$z_1(\xi) \coloneqq \begin{cases} c_1 & \text{if } a_1 \leq \xi \\ 0 & \text{otherwise.} \end{cases}$$

Then, for $\gamma = 2, \ldots, n$, the following recursion can be defined to determine $z_n(b_1)$:

$$z_\gamma(\xi) \coloneqq \begin{cases} z_{\gamma-1}(\xi) & \text{if } a_k > \xi \\ \max\{z_{\gamma-1}(\xi), c_\gamma + z_{\gamma-1}(\xi - a_\gamma)\} & \text{otherwise.} \end{cases}$$

The dynamic programming approach can also be extended to the multi-objective case with $p \geq 2$, i.e., to solve (MO01LP). Then, every state ξ in stage γ contains not just one optimal solution but a set of non-dominated points of the respective subproblem. A detailed description of general multi-objective dynamic programming and problem specific application is given in, for example, Klötzler (1978) and Kostreva and Lancaster (2008). Klamroth and Wiecek (2000) review multiple single-objective dynamic programming approaches available in the literature for knapsack problems. Further, they extend this approaches to the multi-objective case and propose a comprehensive dynamic programming framework.

3.2.2 Multi-objective Branch and Bound

While dynamic programming can only be applied to problems that satisfy the Bellman's Principle of Optimality, multi-objective branch and bound is a generic approach which can be applied to all MOILPs. The general idea is similar to the dynamic programming approach since it is based on a decomposition of the underlying problem into smaller subproblems.

Due to its universal usability the class of decision space methods mostly refers to multi-objective branch and bound algorithms. We give a detailed description of the crucial components and an extensive literature overview.

A multi-objective branch and bound algorithm operates in the same way as its well-known single-objective version. Analogously to the single-objective case, multi-objective branch and bound can also be extended to continuous optimization problems (see, e.g., Eichfelder et al., 2021). However, we restrict ourselves to the description of discrete branch and bound algorithms.

Since the underlying (MO01LP), which needs to be solved, is too hard to solve directly, it is divided into easier subproblems. Every created subproblem is associated with a node, resulting in a tree data structure where the root node represents the original problem. Thereby, a node i is the child node of node j if and only if the fesaible set of the subproblem corresponding to node i is a subset of the feasible set of the (sub)problem corresponding to node j. One of the first, if not the first, multi-objective branch and bound approaches was developed by Klein and Hannan (1982). Although the authors optimize only one of the p objectives and add constraints for the remaining $p-1$ objectives, which resembles objective space methods (cf. Section 3.1), the presented approach has an underlying tree structure. Therefore, it belongs to the class of branch and bound approaches.

In each iteration, an active node is selected and its corresponding lower bound set is computed. We start with the root node to which the original problem is associated. After

the computation of the lower bound set, we possibly update the incumbent list and its corresponding upper bound set and check if it is possible to fathom the node. If the node cannot be fathomed, the corresponding problem needs to be further divided into new subproblems (branching). As a result a branch and bound method is made up of the following components:

- Lower bound
- Upper bound
- Node selection
- Fathoming
- Branching

The basic structure of a multi-objective branch and bound algorithm is summarized in Algorithm 6.

Algorithm 6: Multi-objective Branch and Bound Algorithm

Step 0 Let ν^0 be the node corresponding to the initial problem. Initialize the list of nodes $\mathcal{N} := \{\nu^0\}$ and the incumbent list $X_{\mathcal{U}} := \emptyset$.

Step 1 Select and remove a node ν from $\mathcal{N}$.

Step 2 Compute the lower bound set for the (sub)problem corresponding to ν.

Step 3 If a new integer feasible solution has been found update $X_{\mathcal{U}}$ and $\mathcal{U}$ if necessary.

Step 4 Check whether node ν can be fathomed. If ν can be fathomed and $\mathcal{N} \neq \emptyset$ go to Step 1.

Step 5 Create two disjoint subproblems. Add the corresponding newly created nodes to $\mathcal{N}$. If $\mathcal{N} \neq \emptyset$ go to Step 1.

Since the choice of each component is crucial for the performance of this method, we present some of the most frequently used approaches for each of the components.

Lower bound In a branch and bound approach, in each iteration a node ν is selected from the list of nodes $\mathcal{N}$. Then, the lower bound set is computed for the corresponding problem. Since bounding is considerably weaker in the multi-objective case, the bound computation approach is crucial for the performance of these methods. Hence, different approaches for the computation of the lower bound have been proposed in the last decades.

One of the first approaches was the so-called *minimal completion* (see, e.g., Kiziltan and Yucaoğlu, 1983; Klein and Hannan, 1982). In this approach, each variable is fixed to 0 or 1 (for (MO01LP)) depending on the sign of the corresponding objective coefficient. In the single-objective case this is done by fixing each free variable $x_i, i \in \{1, \dots, n\}$ to 1 if the corresponding objective coefficient $c_i \leq 0$. Thus, x_i is fixed to 0 if $c_i > 0$. In Klein and Hannan (1982), this is done in the same way since the authors consider ε-constraint scalarizations of the subproblems. The approach of Kiziltan and Yucaoğlu (1983) is a straightforward extension to the multi-objective case. Thereby, the variable $x_i, i \in \{1, \dots, n\}$ is fixed to 1 if the sum of the corresponding objective coefficients is smaller or equal 0, i.e., $\sum_{j=1}^{p} c_i^j \leq 0$. Although such a solution is integer it is not necessarily feasible, since it could violate certain constraints. The lower bound set is given by the corresponding point in the objective space.

Another single point bounding approach is to use the ideal point as the lower bound. In contradiction to the minimal completion approach, the ideal point approach considers the constraint structure of the underlying problem. This approach yields in general a better lower bound and is the best possible single point lower bound (cf. Section 2.3). For its computation we need to solve p single-objective integer problems. Since this might result in an expensive computation time, it is often replaced by using the ideal point of the linear relaxation (see, e.g., Mavrotas and Diakoulaki, 1998, 2005). The bound can be computed by the costs of solving p single-objective linear continuous problems, which is in general significantly faster (Vincent et al., 2013).

Since using a single point as the lower bound indicates rather weak bounds, there are several approaches that use bound sets consisting of multiple points (cf. Section 2.3). Sourd and Spanjaard (2008) propose to use the solution of a convex relaxation as lower bound set for bi-objective problems. This convex relaxations based on the convex combination, i.e., the weighted sum, of objective functions can systematically be solved by using, e.g., a dichotomic scheme. A bi-objective dichotomic scheme is presented in Aneja and Nair (1979) and is extended to the multi-objective case in Przybylski et al. (2010a) and Przybylski et al. (2019). Although using the convex relaxation yields a strong lower bound set, there are drawbacks. Its computation tends to be very costly since numerous single-objective integer problems have to be solved. Sourd and Spanjaard (2008) propose an approach to improve the computation time by warmstarting the lower bound computation with the corresponding lower bound set of the parent node. However, Vincent et al. (2013) showed for their tested instances that using that lower bound set produces the fewest number of nodes but requires high computation times. Another crucial aspect is the way this lower bound is computed. Since the lower bound set is obtained by computing supported non-dominated points with an inner approximation, the complete convex relaxation has to be

solved to guarantee a valid lower bound. Thus, it is not possible to stop the computation early and use the already computed bound.

Like in the single-objective case, the most common lower bound set is obtained by solving the linear relaxation. This bound is in general weaker compared to the bound obtained by convex relaxation. However, the computation takes less time since the single-objective continuous problems can be solved with commercial solvers like, e.g., CPLEX or GLPK. In Vincent et al. (2013), Belotti et al. (2013) and Adelgren and Gupte (2022), the linear relaxation is used to compute the lower bound set for mixed integer linear programs and for integer linear problems (MOILP) in, e.g., Parragh and Tricoire (2019) and Gadegaard et al. (2019). In Parragh and Tricoire (2019), the computation of the lower bound set is warmstarted by using information of the parent node. There are different ways to solve the linear relaxation. In the bi-objective case, it can be solved with the parametric simplex (see, e.g., Ehrgott, 2005). Besides using the already discussed dichotomic schemes the lower bound set can also be obtained with *Benson's outer approximation algorithm* (Benson, 1998; Ehrgott et al., 2012). The algorithm is initialized with a (weak) lower bound which is improved in each iteration.

Definition 3.10. *Let $\mathcal{K}, \mathcal{P} \subset \mathbb{R}^p$ be polyhedrons. If $\mathcal{K}_N \subseteq \mathcal{P}$, then $\mathcal{P}$ is called an outer approximation of $\mathcal{K}$.*

Let Y^{LR} be the image in the objective space corresponding to the feasible solutions of the linear relaxation X^{LR} of an underlying problem. Then, the algorithm iteratively generates better outer approximations of the polyhedron $\mathcal{P}^{LR} \coloneqq Y_N^{LR} + \mathbb{R}^p_{\leqq}$. In the basic approach, it is started with a polyhedron $\mathcal{P} \coloneqq \bar{y} + \mathbb{R}^p_{\leqq}$ where $\bar{y}$ is dominating the ideal point, i.e., $\bar{y} \leq z(x)$ for all $x \in X^{LR}$. In each iteration, one of the facets of $\mathcal{P}^{LR}$ is constructed by a hyperplane. An outer approximation $\mathcal{P}$ has to be updated with a cut, defined by a hyperplane, if one of the vertices $v \in V_{\mathcal{P}}$ of the vertex-ray representation of $\mathcal{P}$ is not included in $\mathcal{P}^{LR}$. Therefore, Hamel et al. (2013) proposed to solve the following single-objective problem in order to check if $v \in \mathbb{R}^p$ is in $\mathcal{P}^{LR}$:

$$\begin{aligned} \min \quad & \theta \\ \text{s.t.} \quad & \bar{A}\, x \leqq \bar{b} \\ & C\, x \leqq v + \theta\, c \\ & x \geqq 0 \\ & \theta \geq 0, \end{aligned} \tag{3.5}$$

where $C \in \mathbb{R}^{p \times n}$ is the matrix of objective coefficients, $c = (1, \ldots, 1)^\top \in \mathbb{R}^p$ and $v \in \mathbb{R}^p$ is the vertex that is checked. If the objective value θ of (3.5) is equal to zero, then $v \in \mathcal{P}^{LR}$. Otherwise, we need to compute a cutting plane to separate the polyhedron and vertex v.

In Hamel et al. (2013), it is shown that the dual of (3.5) can be transformed to

$$\begin{aligned} \max \quad & -\bar{b}^\top \psi - v^\top \omega \\ \text{s.t.} \quad & -\bar{A}^\top \psi \leqq C^\top \omega \\ & c^\top \omega = 1 \\ & \psi, \omega \geqq 0. \end{aligned} \tag{3.6}$$

Let ψ^*, ω^* be the optimal solution of (3.6), then the hyperplane $\mathcal{H}(v) \coloneqq \{y \in \mathbb{R}^p \colon \omega^\top y = \bar{b}^\top \psi\}$ separates v from $\mathcal{P}^{LR}$ and defines a facet of $\mathcal{P}^{LR}$. Consequently, the outer approximation can be updated, i.e., $\mathcal{P}^{new} \coloneqq \mathcal{P} \cap \mathcal{H}^+(v)$ with $\mathcal{H}^+(v) \coloneqq \{y \in \mathbb{R}^p \colon \omega^\top y \geq \bar{b}^\top \psi\}$. This procedure is repeated until all vertices of the outer approximation are contained in $\mathcal{P}^{LR}$. The outer approximation approach ensures that the algorithm can be aborted at any time since the produced lower bound set is valid all the time. In Forget et al. (2022b) an approach for warm-starting this outer approximation algorithm is proposed.

Since the single point lower bounds are rather weak and the more complex lower bound sets, obtained by solving a linear or convex relaxation, tend to have higher computational times due to the sometimes large number of facets, using a single hyperplane as the lower bound set has become more popular. Such a hyperplane can be obtained by solving one weighted sum scalarization. This idea is proposed in Stidsen et al. (2014) for bi-objective mixed integer programs and is also used in Stidsen and Andersen (2018). In Parragh and Tricoire (2019) and Gadegaard et al. (2019), this approach is combined with the more complex linear relaxation approach for bi-objective instances. De Santis et al. (2020) propose a method for multi-objective ($p \geq 3$) (mixed) integer programs.

There exist also other approaches to obtain lower bound sets. However, most of them are very problem specific (see, e.g., Jozefowiez et al., 2012). The approaches presented above are the most commonly used techniques to obtain a valid lower bound set and can be applied to general (MOILP).

Upper bound In single-objective branch and bound algorithms, the upper bound is usually given by the best found solution so far. In the multi-objective case, the direct extension of this idea is used. An upper bound set $\mathcal{U}$ is given by an incumbent list $X_{\mathcal{U}}$. During the run of the algorithm all feasible solutions are stored in this incumbent list if their images are not dominated by the image of another feasible solution found so far. In particular, after computing the lower bound set, the solutions corresponding to its extreme points are checked for integer feasibility. Such a feasible solution $\hat{x} \in X$ is added to $X_{\mathcal{U}}$ if there is no other $\bar{x} \in X_{\mathcal{U}}$ dominating it, i.e., $z(\bar{x}) \leq z(\hat{x})$. If the new solution $\hat{x}$ is added to $X_{\mathcal{U}}$

all solutions $x \in X_{\mathcal{U}}$ that are dominated by $\hat{x}$ are removed from the incumbent list.

$$X_{\mathcal{U}} \coloneqq \begin{cases} X_{\mathcal{U}} & \text{if } \exists\, x \in X_{\mathcal{U}} \colon z(x) \leq z(\hat{x}) \\ \{\hat{x}\} \cup \{x \in X_{\mathcal{U}} \colon z(\hat{x}) \not\leq z(x)\} & \text{otherwise.} \end{cases}$$

The set $\mathcal{U}$ is given by $\mathcal{U} \coloneqq z(X_{\mathcal{U}})$ and satisfies the properties of an upper bound set, formulated in Definition 2.10 .

Node selection In each iteration of a branch and bound method, the first task is to select an unexplored node from the tree of subproblems. We call this selected node *active*. Since the order of the considered nodes has a significant impact on the number of created nodes and thus on the computational time, there have been several approaches proposed in the literature. Thereby, we distinguish between *static strategies* and *dynamic strategies*. While dynamic strategies consider information gained in previous iterations for the choice of the active node, static strategies consider the nodes in a consistent order, which might depend on the instance, e.g., on objective or constraint coefficients.

Most of the multi-objective branch and bound algorithms proposed in literature use static strategies. The best-known static strategies are the *depth-first strategy* and the *breadth-first strategy*. The greatest advantage of both strategies is also their greatest shortcoming; they are problem independent. Hence, they can be applied in the same way as in its single-objective counterpart. Nevertheless, they do not take the information obtained in previous iterations into account, which might lead to rather bad node selections. The depth-first strategy is used in the majority of branch and bound approaches (see, e.g., Kiziltan and Yucaoğlu, 1983; Ulungu and Teghem, 1997; Vincent et al., 2013; Visée et al., 1998). In, for example, Parragh and Tricoire (2019) the breadth-first strategy is used after showing that it is more convenient for some of their problem classes. In Vincent et al. (2013), it is shown that the depth-first strategy performs significant better on their randomly generated test instances. Hence, it seems problem dependent which strategy performs better. So it might be promising to use node selection strategies that adapt to the structure of the underlying problem.

Although it is common to use dynamic strategies in single-objective branch and bound methods they are rarely applied in the multi-objective case. Nevertheless, several different dynamic approaches have been proposed in the last decades. One of them is to rely on the optimal objective value of a linear relaxation of a weighted sum scalarization. In the first place, this relaxation is solved to generate the lower bound set of the corresponding subproblem. This technique is used by, e.g., Stidsen et al. (2014), Stidsen and Andersen (2018), Gadegaard et al. (2019). In Belotti et al. (2013), another dynamic strategy is

proposed. For all solutions computed by the linear relaxation (lower bound computation) the integer infeasibility is summed up for all (integer) variables. Then, the node with the largest sum of integer infeasibility (distance to the nearest integer solution) is selected. One of the most used dynamic node selection strategies in the single-objective case is to choose the node with the largest gap between the lower and upper bound. But since the bounds are given by sets with multiple points in the multi-objective case, there are numerous options to measure the gap. In Jesus et al. (2021), the gap is computed with the so-called ε-indicator. Adelgren and Gupte (2022) use a slightly adapted Hausdorff distance as gap measure. Forget and Parragh (2023) compare this gap measurement and a best bound value derived from solving a weighted sum problem. We propose new node selection strategies in Chapter 4 and Chapter 5 for the bi-objective respectively multi-objective case by computing the so-called *approximated hypervolume gap*.

Fathoming In a worst-case scenario, a branch and bound approach produces the total enumeration of all feasible solutions. However, fathoming rules allow to discard parts of the solution space that are not necessary to determine a minimal complete set. The three different cases that might occur are: infeasibility, optimality or dominance.

Fathoming by infeasibility: This case is completely analogous to the single-objective branch and bound. If a relaxation used to determine the lower bound set is infeasible, then the corresponding subproblem is infeasible, as well. This holds true since the feasible set of the subproblem is a subset of the feasible set of its relaxation. Consequently, it is possible to fathom the corresponding node by infeasibility.

Fathoming by optimality: Like in the single-objective case a node can be fathomed by optimality if the upper bound $\mathcal{U}$ is equal to the lower bound $\mathcal{L}$. This implies that this node is solved to optimality and that there is no need to divide it further. But since the bound sets in general consists of multiple points this rarely happens. In the cases where connected lower bound sets are used it is not possible to obtain equality and therefore no nodes can be fathomed by optimality. The only possibility where this rule can be applied is when the lower and upper bound set consist of the same single point, i.e., the ideal point.

Fathoming by dominance: A node can be fathomed by dominance if all points of the lower bound set are dominated by at least one point of the upper bound set, i.e., if the objective vectors of all feasible solutions of this subproblem are dominated by objective vectors of points of the incumbent list. This dominance check might lead to difficulties since the bound sets could possibly have different properties. Although the upper bound

set, i.e., the image of the incumbent list, is always a finite set of points for (MO01LP), the lower bound can be computed in different ways. Therefore, the lower bound set can be composed of a finite set of points (e.g., the ideal point), a unique hyperplane (e.g., obtained by solving a single weighted sum scalarization) or a complex lower bound (e.g., obtained by solving the linear relaxation). Depending on the shape of the lower bound there are different approaches to verify dominance. The dominance check for lower bound sets consisting of a finite number of points is straightforward. Every point of the upper bound set is tested against each point of the lower bound set resulting in a finite number of pairwise comparisons. Since a lower bound set with a finite number of points is required for this dominance test it has been used mostly in the early publications about multi-objective branch and bound approaches (see, e.g., Kiziltan and Yucaoğlu, 1983; Klein and Hannan, 1982; Mavrotas and Diakoulaki, 1998; Visée et al., 1998).

Since in the last decades an increasing number of branch and bound approaches with a more complex lower bound set were proposed, Sourd and Spanjaard (2008) present a more general dominance test. It is possible to fathom a node ν if the objective vectors of all feasible points of the subproblem corresponding to node ν are dominated by an objective vector of a point of the incumbent list. This can be done easily by checking if all local upper bounds are located below the lower bound set. In Sourd and Spanjaard (2008) and Gadegaard et al. (2019), it is shown that this dominance test is valid for lower bound sets that are obtained by solving the linear relaxation or the convex relaxation.

Another dominance test for bi-objective (MOILP) is used in Parragh and Tricoire (2019). This test was proposed in Vincent et al. (2013) for bi-objective mixed-integer linear programs. The idea is to compute the lower bound $\mathcal{L}$ and compare it to $\mathcal{U}$. Then, the parts of the lower bound that are dominated by $\mathcal{U}$ are discarded, while the remaining parts of the lower bound set are kept as disjunct subproblems. If no parts have to be kept, the corresponding node can be fathomed by dominance. This procedure is also called *objective branching*, which is described in more detail in the following paragraph "branching". Note that this can lead to many subproblems which can be partially handled by the introduction of so-called *super local upper bounds* (Forget et al., 2022a).

By considering lower bound sets which are composed by an unique hyperplane, it is also possible to use the dominance test of Sourd and Spanjaard (2008). However, the dominance test can be eased in practice like it is shown in Stidsen et al. (2014). The authors propose to keep track of the largest weighted sum value of the solutions in the incumbent list and compare it to the objective value of the weighted sum scalarization which is solved to obtain the lower bound. So, the dominance test is simplified to a comparison of two single values. Additionally, the objective space is partitioned in different slices that are handled independently. This is also an objective branching method.

An overview of existing rules for fathoming by dominance can be found in Belotti et al. (2016).

Branching If a node cannot be fathomed its corresponding problem is further divided into smaller (usually disjoint) subproblems. Since we are considering (MO01LP) the two subproblems are created by fixing a free variable x_i to 0 or to 1 respectively. Thus, the constraint $x_i = 1$ or $x_i = 0$ is added to the corresponding subproblem. Thereby, $i \in \mathcal{I}(\nu) \subseteq \{1, \dots, n\}$, where $\mathcal{I}(\nu)$ is the set of free variables of the corresponding node ν, i.e., $\mathcal{I}(\nu)$ contains the indices of the variables which have not been fixed yet. This procedure results in a binary tree structure and is called decision space branching.

Of course, the choice of the variable on which the branching is applied is crucial for the performance of the branch and bound algorithm. Therefore, several branching rules have been proposed. Therefore, we distinguish again between static and dynamic strategies. If the order of the variables that are branched is known in advance, we call this strategy static. In every iteration, the first variable of the list of free variables of the the corresponding subproblem is selected. In Achterberg et al. (2005), a comparison of the performance of different branching strategies is given for the single-objective case.

The most basic static strategy is to always choose the free variable with the smallest/largest index (see, e.g., Mavrotas and Diakoulaki, 1998). This approach is problem independent and therefore, its impact is based on luck. For the single-objective case, Kellerer et al. (2004) propose a static branching rule for knapsack problems by choosing the most promising variable according to the profit to weight ratio value c_i/a_i, $i \in \{1, \dots, n\}$, where $c \in \mathbb{R}^n$ is the objective vector and $a \in \mathbb{R}^n$ contains the constraint coefficients. Although there is no direct extension to the multi-objective case, there are some approaches that consider the conflicting objective function coefficients (see, e.g., Bazgan et al., 2009; Jorge, 2010; Ulungu and Teghem, 1997). In Chapter 6, we show how this technique can also be adapted to other problem classes. Vincent et al. (2013) propose a similar static branching rule based on the objective functions for the bi-objective case where the variables are ranked based on their objective coefficients which makes it applicable to every problem class.

Dynamic branching strategies take information into account that has been gained in previous steps of the branch and bound algorithm and rely the variable selection on it. Although most of the published papers use static strategies, there are several dynamic approaches. One of them is to count how often a variable is not integer in all solutions corresponding to the extreme points of the computed lower bound set and then to choose the variable, which is most often fractional. Another approach is to consider the L_1-distance to the nearest integer solution and choose thus the variable that is most fractional

(Belotti et al., 2013). Stidsen et al. (2014), Stidsen and Andersen (2018) and Gadegaard et al. (2019) give a single-objective solver the choice of their next variable to branch on similar to their node selection strategies. Note that this is can be done due to the way the lower bound is computed.

Additional to the classical decision space branching a method to create subproblems has arose in the recent years. Instead of dividing the decision space by variable fixing, the objective space is partitioned by adding constraints on the objective functions. This procedure is often referred to as *objective branching*. Stidsen et al. (2014) propose two different techniques to subdivide the objective space. In the first one, the objective space is partitioned into different slices in the bi-objective case. Each of this created slices is evaluated independently, which also makes it possible to parallelize the underlying branch and bound (Stidsen and Andersen, 2018). In a second approach proposed by Stidsen et al. (2014), upper bounds on the objective functions are added to discard parts of the objective space where the lower bound is already dominated by points of the upper bound set. In practice, for each part of the lower bound that is not dominated by $\mathcal{U}$ a new subproblem is created. This approach was improved by Gadegaard et al. (2019) and Parragh and Tricoire (2019). Although this approaches show promising results for the bi-objective case, for a long time there was no extension to the multi-objective case. Forget et al. (2022a) propose an objective branching algorithm for three and more objectives.

In the following chapters, we present different improvements and augmentations to some of the key components of multi-objective branch and bound. By referring to the basic structure, given in Algorithm 6, we can specify at what stage of the algorithm the new approaches are applied. Additionally, we define a basic multi-objective branch and bound framework by describing for each key component which approach is used. The presented augmentations are then applied to this basic framework, which is given in the following.

Basic Multi-objective Branch and Bound Framework

- *Lower bound:* linear relaxation
- *Upper bound:* incumbent list
- *Node selection:* depth-first strategy
- *Branching rule:* most often fractional

A survey on multi-objective branch and bound approaches is provided in Przybylski and Gandibleux (2017), a survey for its single-objective counterpart can be found in Morrison et al. (2016).

4 Augmenting Bi-objective Branch and Bound by Scalarization-Based Information

Although in the single-objective case the branch and bound based algorithms are the gold standard for integer programming problems, objective space methods, presented in Section 3.1, are preferred in the multi-objective case. This is mainly due to the fact that objective space methods benefit from optimized single-objective solvers to solve scalarized subproblems, while branch and bound methods suffer from the considerably weaker bounding in multiple objectives (cf. Section 2.3).

In this chapter, we focus on bi-objective integer optimization problems and present bi-objective branch and bound algorithms that are augmented by scalarization-based information. We propose modifications which improve the computational efficiency of bi-objective branch and bound algorithms in two critical aspects. One of the weaknesses of multi-objective branch and bound as compared to its single-objective counterpart is the bounding procedure. While any feasible solution $\bar{x} \in X$ dominates w.r.t. one (linear) objective function a half-space in decision space (i.e., $\{x \in \mathbb{R}^n \colon c^\top x \geq c^\top \bar{x}\}$), the set of feasible solutions which are dominated by a solution $\bar{x}$ in $p \geq 2$ objective functions ($C \in \mathbb{R}^{p \times n}$) forms a cone $\{x \in \mathbb{R}^n \colon C\,x \geqq C\,\bar{x}\}$. The cone of dominated solutions is smaller the more the objective functions are in conflict, leading also to a larger number of efficient solutions. This implies that a significant part of the branch and bound tree has to be enumerated and only a small number of branches can be pruned by dominance. Despite of this general problem in multi-objective optimization, this asks for good bounding procedures to avoid the unnecessary evaluation of dominated branches. This however, requires good solutions in the incumbent list in early stages of the algorithm as well as tight lower bounds.

In order to achieve this, we suggest a new branching strategy and the hybridization of branch and bound with objective space methods. We determine scalarized subproblems adapted to the state of the branch and bound and solve these to integer optimality. During the run of branch and bound algorithms, scalarized relaxations are frequently solved to compute the lower bound set. Since our approach involves additionally the solution of scalarizations to integer optimality we refer to them as *IP scalarizations*.

Contribution Most of this chapter is contained in Bauß and Stiglmayr (2024b).

Organization of the Chapter The remaining chapter is organized as follows. In Section 4.1, we propose a new dynamic node selection strategy which follows a best-first approach. In Section 4.2, several approaches to improve the bound sets with IP scalarizations are introduced. Furthermore, two different kinds of hybrid bi-objective branch and bound algorithms are proposed. Numerical results of different combinations of the presented approaches are presented in Section 4.3, which show their impact and effectiveness.

4.1 A New Bi-objective Branching Strategy

The branching strategy comprises two subsequent decisions: the choice of the active node and its branching into subproblems, i. e., the decision on which variable the subproblem is branched. This second step is denoted as branching rule. We discuss these two steps together since the order in which the nodes are considered has a significant impact on the branched variable. Instead of the static depth- or breadth-first we use a dynamic node selection strategy, while we rely on the most fractional rule as branching rule.

The basic idea of our strategy is quite simple and a natural extension of choosing the largest gap in the single-objective case (see, for example, Dechter and Pearl, 1985). For every created node we compute the *approximate hypervolume gap* between lower and upper bound. We use the definition of hypervolume proposed in Zitzler and Thiele (1999). In every iteration, we choose the node with the largest hypervolume gap as active node (cf. Jesus et al., 2021). Note that when a node is created during the branching process, the approximated hypervolume gap of the parent node is assigned to it. We distinguish two variants of the hypervolume gap: the *total hypervolume gap* and the *local hypervolume gap*. While the total hypervolume gap measures the volume of the search region, i. e., the volume between lower and upper bound set, the local hypervolume gap approach considers only the volume of the largest search zone, i. e., the gap between a local upper bound and the lower bound set. For a more detailed definition of search regions and search zones we again refer to Klamroth et al. (2015).

Figure 4.1 illustrates the two different approaches. Here, $z^1, \dots, z^4 \in \mathcal{U}$ are points of the upper bound and $lu^1, \dots, lu^3 \in K \subseteq \mathcal{D}(\mathcal{U})$ are their corresponding local upper bounds, where K is a subset of the local upper bounds containing just the points above the lower bound of node $\bar{\nu}$. The blue line represents the lower bound. Figure 4.1(a) shows how to measure the total hypervolume gap of a node $\bar{\nu}$, in the following denoted by $thg(\bar{\nu})$. For this approach we consider the approximated search region of the corresponding node. Since there is a natural order in the bi-objective case, it is possible to consider the approximated search zone of the first local upper bound, i.e., the local upper bound with

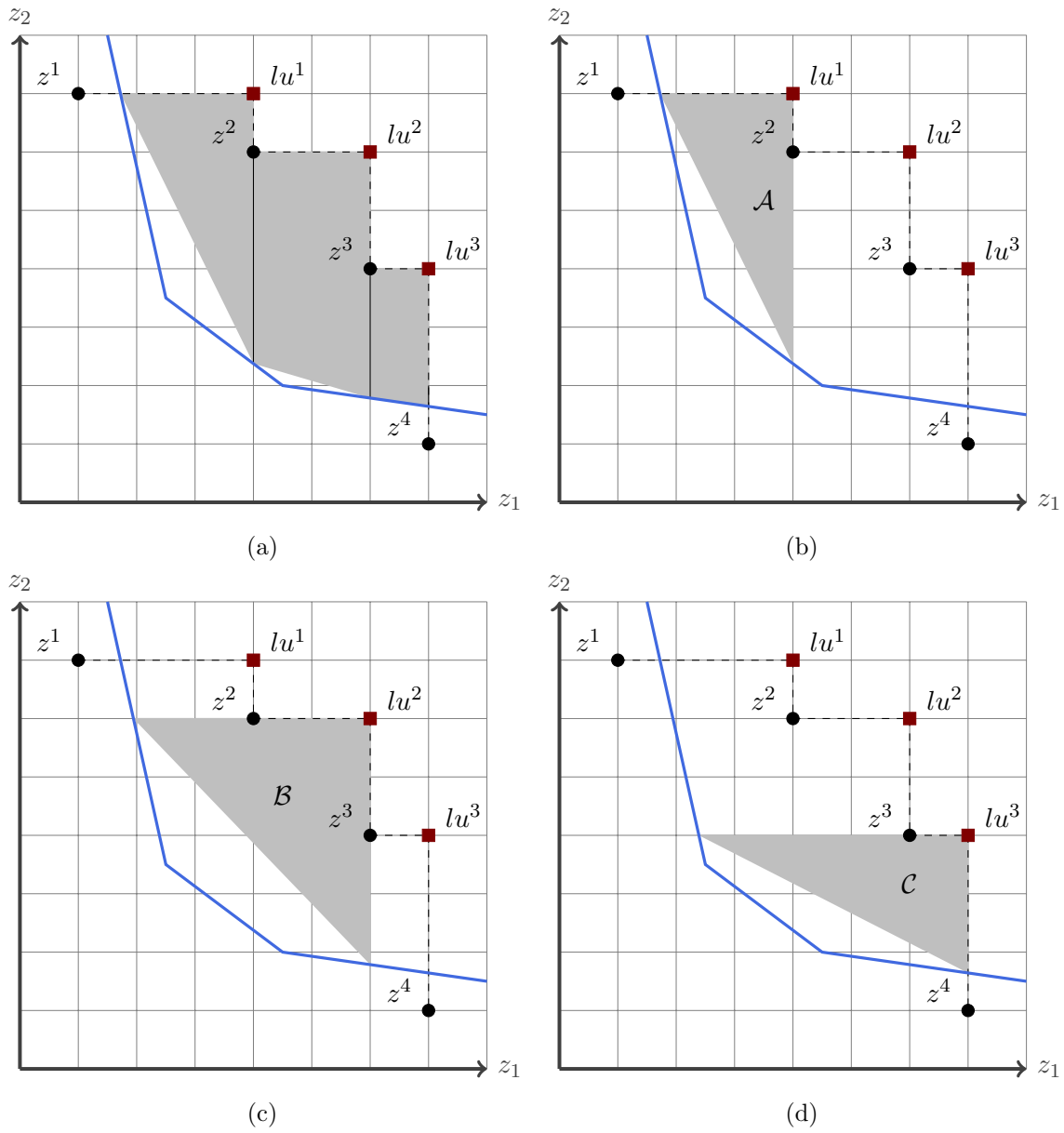


Figure 4.1: Example of computation of the two different approximated hypervolume gap approaches.

the smallest z_1-value. Therefore, we define the two spanning points, which, together with the corresponding local upper bound, define a triangle. The spanning points of a local upper bound lu are defined by $sp^i(lu) \coloneqq \{l \in \mathcal{L} \colon l_{3-i} = lu_{3-i}\}, i = 1, 2$. So, the approximate hypervolume gap of lu is given by

$$hg(lu) \coloneqq \frac{1}{2} \left| sp^1(lu)_1 - lu_1 \right| \cdot \left| sp^2(lu)_2 - lu_2 \right|.$$

For the remaining local upper bounds we compute the hypervolume of slices as shown in the illustration. The hypervolume of the slice of $lu^i, i = 1, \ldots, |K|$ is defined as

$$sl(lu^i) := \frac{\left|z_2^i - sp^2(lu^{i-1})_2\right| + \left|lu_2^i - sp^2(lu^i)_2\right|}{2} \cdot \left|z_1^i - lu_1^i\right|.$$

So, the total (approximated) hypervolume gap, which is assigned to node $\bar{\nu}$, is given by

$$thg(\bar{\nu}) := hg(lu^1) + sl(lu^2) + \ldots + sl(lu^{|K|}).$$

The Figures 4.1(b), 4.1(c) and 4.1(d) show the computation of the local hypervolume gap. The local hypervolume gap of a node $\bar{\nu}$ is considered as the largest approximated hypervolume gap of a local upper bound $lu \in K$. Therefore, the the local hypervolume gap, which is assigned to node $\bar{\nu}$, is defined by

$$lhg(\bar{\nu}) := \max_{i=1,\ldots,|K|} hg(lu^i).$$

In the given example, $\mathcal{B}$ is the largest approximated hypervolume and therefore is assigned to node $\bar{\nu}$.

In Algorithm 6, the node with the largest assigned hypervolume gap is selected in Step 1. The value of the hypervolume gap is updated in Step 4 if the node cannot be fathomed.

Note that in our presented algorithms in Section 4.2 the local upper bound is initialized with the point $(M, M)^\top$ with a sufficient large value $M \gg 0$. Therefore, it is possible to apply the new branching strategies immediately at the beginning of the algorithm. Obviously, this approximation may neglect significantly large parts of the search regions and search zones. However, the idea of the approximated hypervolume gap eases computation and saves time. The efficiency of these new dynamic branching strategies is shown in Section 4.3.

4.2 Augmenting Bi-objective Branch and Bound by Solving IP Scalarizations

In this section, we introduce a method to incorporate scalarizations into branch and bound. We build a hybrid branch and bound algorithm combining the partial enumeration of decision space with objective space information by solving scalarizations to integer optimality.

An integer optimal solution $\bar{x}$ of a scalarization can be used to update upper and lower bound. Obviously, the solution $\bar{x}$ can be added to the incumbent list and its corresponding image point $z(\bar{x})$ can be added to the upper bound set. Moreover, a scalarizing function

and its optimal solution $\bar{x}$ define a level set, which can be included in the lower bound set for all descendant nodes. In order to utilize these improved lower bounds in all nodes we solve the IP scalarizations in the root node.

4.2.1 Using Weighted Sum Scalarizations

During the run of the branch and bound algorithm, a strategy triggers the IP solution of weighted sum scalarizations in the root node. Thus, we solve problem (WS_λ) for for adaptively chosen values of $\lambda \in \Lambda$. Although we solve the IP scalarization in the root node the parameter λ is gained from the currently active node. Thereby, λ is determined by the largest approximated local hypervolume gap in the active node. This gap is spanned by two points in the incumbent list together with their local upper bound. Note that these points spanning the largest gap are already determined if the local hypervolume gap branching strategy is applied. The corresponding value of λ is determined by computing the normal to the hyperplane that is defined by those two points. Once λ is obtained, we can solve problem (WS_λ) with a single-objective integer linear programming solver. Let $\bar{x}^\lambda$ be the optimal solution of the weighted sum scalarization with weighting vector λ, then $z(\bar{x}^\lambda)$ is a supported non-dominated point of (MO01LP) with $p = 2$. Thus, we can add $\bar{x}^\lambda$ to the incumbent list and its corresponding image to $\mathcal{U}$ (if it was not found in previous iterations) and filter the resulting list for non-dominance.

Moreover, the solution of integer scalarizations can also be used to tighten the lower bound set, since the level set $\{y \in \mathbb{R}^2 \colon \lambda^\top y = \lambda^\top z(\bar{x}^\lambda)\}$ provides the valid inequality $\lambda^\top z(x) \geq \lambda^\top z(\bar{x}^\lambda)$ for all $x \in X$.

Figure 4.2 illustrates the update of the lower and upper bound set. In Figure 4.2(a), $z^1, \ldots, z^4$ indicate points that are currently in the upper bound $\mathcal{U}$ and $lu^1, \ldots, lu^3$ are the corresponding local upper bounds. The point $z(\bar{x}^\lambda)$ is obtained by solving a weighted sum scalarization (WS_λ) to integer optimality. Since $\bar{x}^\lambda$ is not contained in the incumbent list so far, we can update the upper bound as it is shown in Figure 4.2(b). The new upper bound then reads as $\mathcal{U} \coloneqq \{z(\bar{x}^\lambda)\} \cup \{z \in \mathcal{U} \colon z(\bar{x}^\lambda) \nleqq z\}$. Moreover, the lower bound set $\mathcal{L}$ can be updated by integrating the green hyperplane into the lower bound set, i. e., $\mathcal{L} \coloneqq \{z \in \mathcal{L} + \mathbb{R}^2_{\geqq} \colon \lambda^\top z \geq \lambda^\top z(\bar{x}^\lambda)\}_N$ as it is shown in Figure 4.2(c) and 4.2(d). In this situation, both —the lower and upper bound— are updated, which is not the case in general.

The example illustrates the benefits of hybridizing multi-objective branch and bound with IP scalarizations. Due to weak bounding, nodes may not be fathomed by dominance even if they do not contain additional non-dominated points. The tighter upper bound increases the probability of fathoming a node by dominance in later iterations of the algo-

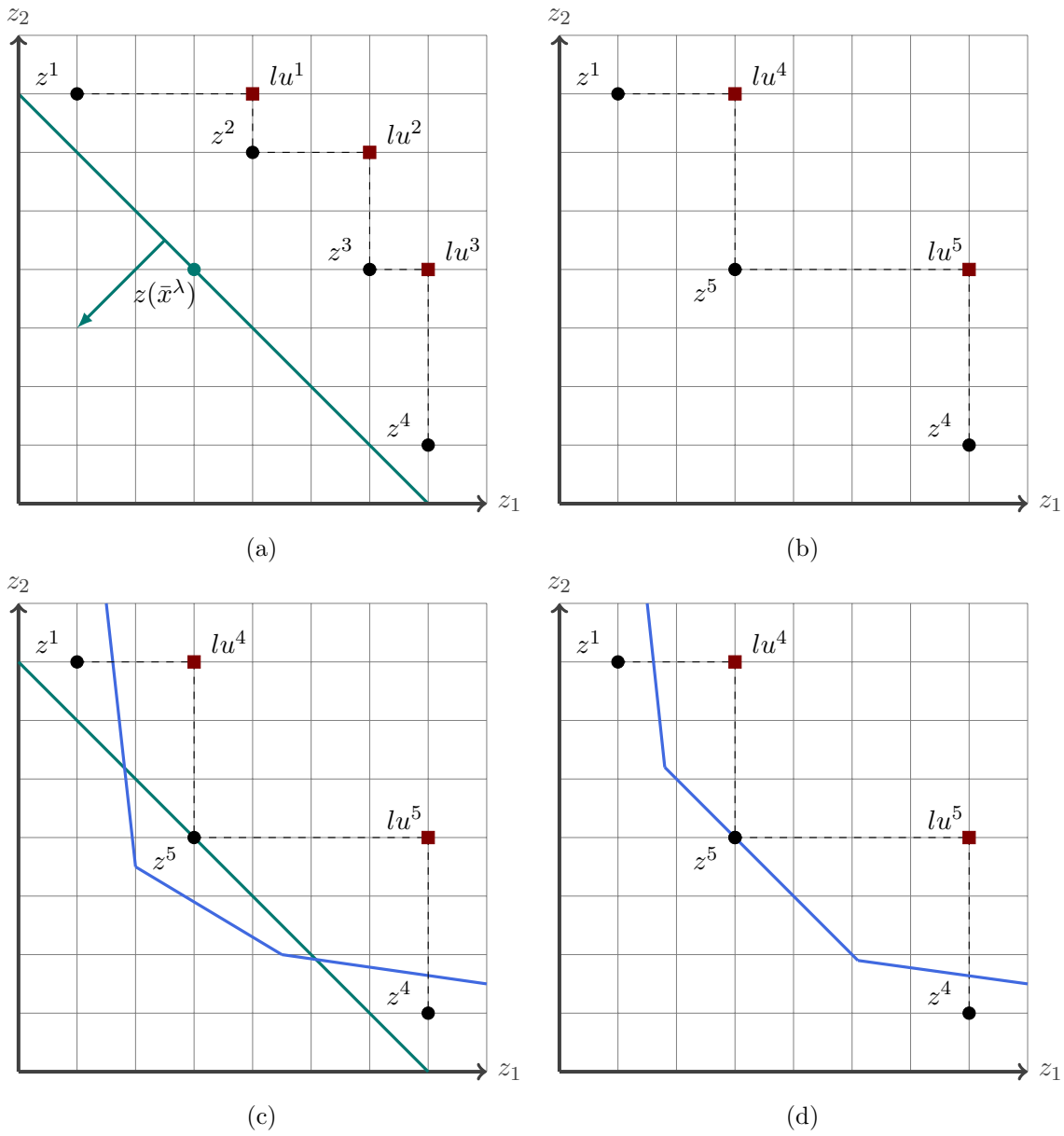


Figure 4.2: A bi-objective example of updating the lower and upper bound with the usage of the weighted sum scalarization.

rithm. Also, the lower bound might be improved. Since we are solving an IP scalarization in the root node, the obtained optimal level set is a valid inequality for all subproblems.

The weighted sum scalarization is applied between Step 2 and Step 3 of Algorithm 6.

We combine our new branching strategy and the augmentation with IP scalarizations to our first hybrid branch and bound approach.

Hybrid Bi-objective Branch and Bound Algorithm using Weighted Sum Scalarization

- *Lower bound:* linear relaxation
- *Upper bound:* incumbent list
- *Node selection:* node with the largest total/local hypervolume gap
- *Branching:* most fractional rule
- Adaptively solve weighted sum scalarizations in the root node to integer optimality to improve lower and upper bounds by objective space information

Instead of using a static depth-first strategy (as in the basic multi-objective branch and bound framework presented in Section 3.2.2) we apply the dynamic strategy based on the hypervolume gap (c.f. Section 4.1). Even though the extreme points of the lower bound sets might be updated by the weighted sum scalarization, the branching variable is selected based on the original lower bounds. This is due to the fact that the preimages of such intersection points of IP scalarizations and the lower bound set are in general not available. Note that the weighted sum IP scalarizations are included *adaptively* into the branch and bound. The description of their algorithmic control, however, is postponed to Section 4.2.3.

In order to conclude the description of the proposed hybrid bi-objective branch and bound algorithm using weighted sum scalarizations, we want to briefly discuss its advantages and shortcomings. Firstly, it is easy to determine the scalarization parameter λ and to integrate the hyperplane into the lower bound set. Its advantage, however, is that the lower bound remains convex. Therefore, the check for fathoming by dominance remains intuitive. Unfortunately, the weighted sum scalarization can only find supported non-dominated points and the lower bound cannot be improved beyond the convex hull of Y_N. This motivates us to consider the augmented weighted Tchebycheff scalarization, a scalarization approach which can determine also unsupported non-dominated points.

4.2.2 Using Augmented Weighted Tchebycheff Scalarizations

To solve an augmented weighted Tchebycheff scalarization (AWT_τ^w), presented in Section 3.1, it is necessary to predefine a reference point s, weights $w \in \mathbb{R}^2_>$ and an appropriate parameter τ. Since we are in the bi-objective case and an appropriate choice for τ is difficult in general, we use the proposed adaptive method of Dächert et al. (2012) to determine the parameters w_1, w_2 and τ.

As a reference point s we use the local ideal point of two adjacent non-dominated points. Since the augmented weighted Tchebycheff scalarization can only determine non-

dominated points (and the corresponding efficient solutions) which are (strictly) dominated by the reference point, we obtain a non-dominated point in this box.

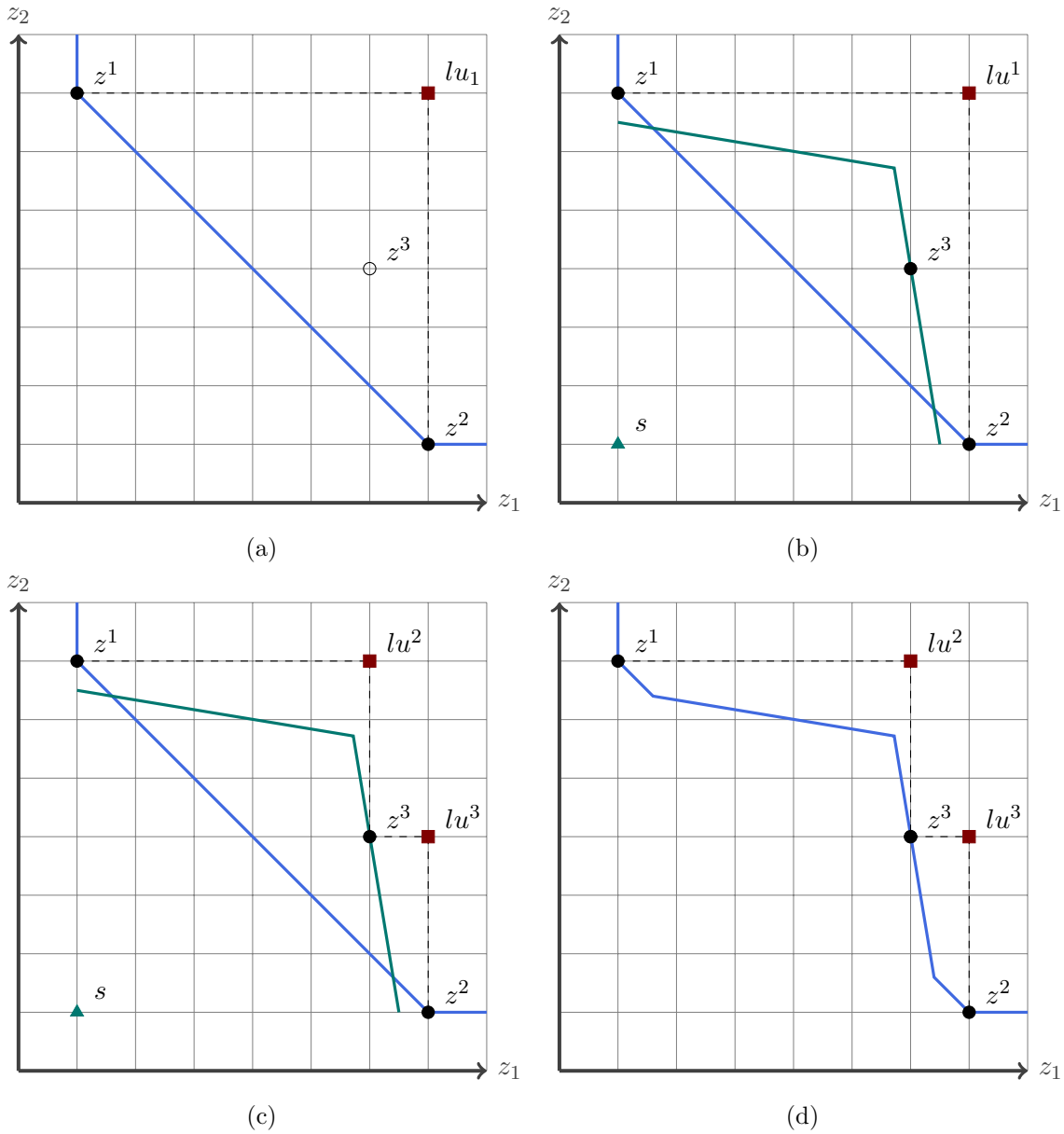


Figure 4.3: A bi-objective example of updating the lower and upper bound with the usage of the augmented weighted Tchebycheff scalarization.

The goal to improve the lower bound set beyond the convex hull of non-dominated points is the motivation to solve augmented weighted Tchebycheff scalarizations to integer optimality. Figure 4.3 shows an example how such an update of the bounds could look like. Here, z^1 and z^2 are two known non-dominated points (obtained with the weighted

sum IP scalarization). Point z^3 is an unsupported non-dominated point that has not been found yet in Figure 4.3(a). By using the local ideal point of z^1 and z^2 as the reference point s, Figure 4.3(b) illustrates how the non-dominated point z^3 is found by applying the augmented weighted Tchebycheff scalarization. In Figure 4.3(c) and 4.3(d), the resulting improvements of the lower and upper bound are shown. Obviously, the lower bound is improved beyond the convex hull of Y_N.

The augmented weighted Tchebycheff scalarization is applied between Step 2 and Step 3 of Algorithm 6. With these modified components we define our second hybrid branch and bound approach:

Hybrid Bi-objective Branch and Bound Algorithm using Augmented Weighted Tchebycheff Scalarization

- *Lower bound:* linear relaxation
- *Upper bound:* incumbent list
- *Node selection:* node with the biggest total/local hypervolume gap
- *Branching:* most fractional rule
- Adaptively solve weighted sum and augmented weighted Tchebycheff scalarizations in the root node to integer optimality to improve lower and upper bounds by objective space information

In addition to the weighted sum scalarization, we use the augmented weighted Tchebycheff scalarization. Since two adjacent non-dominated points are required as input of the augmented weighted Tchebycheff scalarization, we cannot rely on points in the upper bound set, which are only non-dominated so far. In fact, we apply augmented weighted Tchebycheff IP scalarizations only to boxes spanned by points obtained as optimal solutions of the weighted sum scalarization. Thus, we do not rely on parameters from the currently active node, but solve the augmented weighted Tchebycheff scalarization in the largest area defined by two adjacent known non-dominated points.

When using augmented weighted Tchebycheff IP scalarizations, the lower bound can become tighter than the convex hull of the set of non-dominated points, which reduces the area where new non-dominated points can be found. Additionally, we can find unsupported non-dominated points in early stages of the algorithm. This improves the upper bound in the beginning resulting in a higher chance of fathoming a node by dominance. However, this also implies that the lower bound gets non-convex in general, which makes the fathoming tests significantly harder, and the lower bound improves only locally.

4.2.3 Algorithmic Control of IP Scalarizations

In the previous subsections, we did not specify when to solve IP scalarizations, which implies a significant computational cost itself. However, this might be the most crucial part within the presented methods. Obviously, we aim at gaining as much information as possible by solving IP scalarizations. More objective space information will lead to tighter bounds that reduce the number of created nodes, due to a higher probability of fathoming by dominance and smaller search zones. Moreover, a reduced number of created nodes will reduce the total computation time. At the same time, solving overly many IP scalarizations will have a negative impact on the computation time. Furthermore, at a certain point the lower and upper bound will not improve anymore when solving additional IP scalarizations.

So, there exists a trade-off between the reduction of the number of created subproblems and the decrease of the computation time. The difficulty is to find an appropriate condition to trigger an IP scalarization. Obviously, solving IP scalarizations more frequently in the beginning of the branch and bound algorithm is very promising. The earlier the lower and upper bounds are improved the more nodes might be fathomed. Moreover, solving an IP scalarization when the active node has weak bounds will lead to stronger improvements than in later stages of the algorithm. This is complemented by our adaptive branching strategy, which tends to select subproblems with weak lower bounds first.

The hybrid branch and bound algorithm using augmented weighted Tchebycheff scalarization entails also another problem. The augmented weighted Tchebycheff scalarization improves the lower bound just locally. If we use this scalarization at the beginning of the algorithm instead of the weighted sum scalarization, this could lead to an increase of created nodes. Once again, the intuitive idea is to start with the weighted sum IP scalarization more frequently in the beginning of the algorithm. This ensures that the lower bound improves globally at early stages of the branch and bound. The augmented weighted Tchebycheff scalarization should be used in later stages of the algorithm to find unsupported non-dominated points and to improve the lower bound locally. The efficiency of this idea and other approaches will be shown in the next section where we present numerical test results.

4.3 Numerical Tests

All algorithms were implemented in Julia 1.7.1 and the linear relaxations were solved with Bensolve 2.1 (Löhne and Weißing, 2017). The numerical tests were executed on a single core of a 3.20 GHz Intel® Core™ i7-8700 CPU processor in a computer with 32 GB

RAM, running under openSUSE linux Leap 15.3.

We present numerical results of our new approaches and compare them to the basic branch and bound framework presented in Section 3.2.2 which we use as baseline implementation. We consider three different types of problems: multidimensional knapsack problems, assignment problems and discrete uncapacitated facility location problems. The implementation of the proposed multi-objective branch and bound method and the considered benchmark instances are publicly available (Bauß and Stiglmayr, 2023b). Multiple combinations of parameter settings are used to solve these test problems. Thereby, we compare the average number of explored nodes, the average number of solved IPs and the average computation time for 20 instances per problem size. The different evaluated approaches are

- the generic bi-objective Branch and Bouch (BB),
- bi-objective branch and bound using the local (BS1) respectively global (BS2) hypervolume gap as node selection criterion,
- hybrid branch and bound including weighted sum IP scalarizations (WS), and
- different combinations of the hybrid branch and bound algorithm using weighted sum IP scalarization (M1.α.β) and hybrid branch and bound algorithm using weighted sum and augmented weighted Tchebycheff IP scalarization (M2.α.β.γ).

The parameter $\alpha \in \{1, 2, 3\}$ controls how often IP scalarizations are applied. Since the number of IP scalarizations is chosen depending on the problem class, the meaning of the different values for α is described in detail in the corresponding subsections. In general, however, the larger the parameter α is chosen, the fewer IP scalarizations are solved. With β we distinguish between the local ($\beta = 1$) and the global ($\beta = 2$) hypervolume gap strategy. In the hybrid branch and bound algorithm using augmented weighted Tchebycheff scalarization, we also distinguish between integrating the objective space information of the augmented weighted Tchebycheff into the lower bound ($\gamma = 1$) or not ($\gamma = 2$).

Note that the parameter values for each of the problem classes yield from preliminary test runs on a different set of instances, where they shown to provide good results. Thus, the parameter values are chosen problem dependent but are not optimized for the specific test instances.

4.3.1 Bi-objective Multidimensional Knapsack Problems

We consider bi-objective, multidimensional knapsack problems with one, two and three linear restrictions (i. e., $m = 1, 2, 3$). For every problem size we randomly generate 20

instances of the form

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} c_i^j \, x_i && j = 1, 2 \\ \text{s.t.} \quad & \sum_{i=1}^{n} w_i \, x_i \leq b \\ & \sum_{i=1}^{n} \bar{w}_{it} \, x_i \leq d_t && t = 1, ..., m-1 \\ & x \in \{0,1\}^n \end{aligned}$$

with $c_i^j \in [50, 100]$, $w_i \in [5, 15]$, $b = 5\,n, \bar{w}_{it} \in [5, 15]$ and $d_t = \left\lfloor \frac{r\,n}{2} \right\rfloor$ with $r \in [5, 15]$. Depending on the parameter α we specify when and how often IP scalarizations are solved. In M1.1.β and WS, we apply the weighted sum scalarization every 10-th iteration. In M1.2.β, we apply it every 10-th iteration but only within the first n^2 iterations. In M1.3.β, we apply the weighted sum scalarization every 10-th iteration within the first $n^2/3$ iterations, every n-th iteration within the next $n^2/3$ iterations and every $2n$-th iteration within the third $n^2/3$ iterations. In M2.1.$\beta.\gamma$, we apply the weighted sum scalarization every 10-th iteration and every 50-th iteration the augmented weighted Tchebycheff scalarization is used instead. In M2.2.$\beta.\gamma$, we operate like in M1.2.β but after the first n^2 iterations we apply the augmented weighted Tchebycheff scalarization every 50-th iteration. In M2.3.$\beta.\gamma$, we operate like in M1.3.β but after the first n^2 iterations we apply the augmented weighted Tchebycheff scalarization every 50-th iteration. If a scalarization cannot be applied or the same IP scalarization has already been solved before, no IP scalarization is solved in that iteration.

First of all, we notice that our branching strategies have a huge impact on the number of explored nodes and the computation time in knapsack problems. We observe that in general the local hypervolume gap strategy works better than the global hypervolume gap strategy. With the local strategy we can reduce the number of explored nodes by up to 76% (Table 4.1d and Table 4.2d) and the computation time by up to 73% (Table 4.1d). Although the local strategy works better the global hypervolume gap strategy has also a significant impact. The number of explored nodes can be reduced by up to 58% (Table 4.3b) and the computation time by up to 52% (Table 4.3b). The number of nodes and the computation time is reduced in all our approaches and we can notice that combinations with the local hypervolume gap strategy work better.

By limiting the number of solved weighted sum IPs (i.e., in M1.2.β, M1.3.β, M2.2.$\beta.\gamma$ and M2.3.$\beta.\gamma$) we notice two consequences. The number of nodes increases while the number of solved IPs decreases. Although the number of nodes (and thus the number of

knapsack problem, $m = 1, n = 25$				
version	nodes	time (s)	IPs	solved
BB	2701.6	0.8190	0.0	20
BS1	1732.4	0.5392	0.0	20
WS	1539.4	0.7240	10.5	20
M1.1.1	1444.0	0.6850	10.6	20
M1.2.1	1444.0	0.5998	7.2	20
M1.3.1	1452.7	0.5766	5.8	20
M2.1.1.1	1362.5	0.8095	14.1	20
M2.2.1.1	1405.5	0.7378	10.4	20
M2.3.1.1	1423.5	0.6863	7.8	20
M2.1.1.2	1357.0	0.7250	14.0	20
M2.2.1.2	1418.7	0.6586	9.7	20
M2.3.1.2	1439.0	0.6073	6.9	20
BS2	1719.3	0.5389	0.0	20
M1.1.2	1435.3	0.6847	10.1	20
M1.2.2	1438.9	0.6094	6.9	20
M1.3.2	1441.7	0.5788	5.9	20
M2.1.2.1	1363.5	0.7772	13.6	20
M2.2.2.1	1413.5	0.6772	9.0	20
M2.3.2.1	1420.4	0.6408	7.5	20
M2.1.2.2	1358.8	0.7478	14.4	20
M2.2.2.2	1415.3	0.6566	9.4	20
M2.3.2.2	1418.8	0.6281	8.1	20

(a) Knapsack problem with $m = 1$ constraint and $n = 25$ variables.

knapsack problem, $m = 1, n = 50$				
version	nodes	time (s)	IPs	solved
BB	27916.3	18.153	0.0	20
BS1	11788.1	8.339	0.0	20
WS	14270.7	10.507	33.8	20
M1.1.1	10789.7	8.452	26.4	20
M1.2.1	10793.5	8.188	21.2	20
M1.3.1	10795.7	8.116	18.0	20
M2.1.1.1	9888.5	10.873	48.7	20
M2.2.1.1	10140.3	9.437	32.7	20
M2.3.1.1	10521.0	8.774	25.7	20
M2.1.1.2	9840.1	8.396	45.6	20
M2.2.1.2	10130.6	8.422	32.4	20
M2.3.1.2	10401.8	8.288	26.3	20
BS2	16739.8	11.397	0.0	20
M1.1.2	11026.3	8.861	26.1	20
M1.2.2	11024.5	8.860	19.9	20
M1.3.2	11047.4	8.645	16.1	20
M2.1.2.1	10071.8	10.907	45.9	20
M2.2.2.1	10421.4	9.587	31.8	20
M2.3.2.1	10583.2	9.448	24.5	20
M2.1.2.2	9994.1	8.940	46.7	20
M2.2.2.2	10413.4	8.820	32.6	20
M2.3.2.2	10568.9	8.727	25.2	20

(b) Knapsack problem with $m = 1$ constraint and $n = 50$ variables.

knapsack problem, $m = 1, n = 80$				
version	nodes	time (s)	IPs	solved
BB	153938.9	186.330	0.0	20
BS1	36392.0	50.952	0.0	20
WS	58825.7	79.545	54.0	20
M1.1.1	34337.7	50.431	41.7	20
M1.2.1	34333.9	50.312	33.1	20
M1.3.1	34307.1	50.505	26.4	20
M2.1.1.1	31643.7	81.625	100.2	20
M2.2.1.1	32708.9	68.939	76.2	20
M2.3.1.1	32986.3	69.848	63.6	20
M2.1.1.2	31274.5	46.721	102.9	20
M2.2.1.2	32795.7	48.576	76.3	20
M2.3.1.2	33025.8	48.358	63.4	20
BS2	90976.0	116.847	0.0	20
M1.1.2	39745.1	59.321	45.3	20
M1.2.2	40083.2	59.350	31.2	20
M1.3.2	39918.1	58.999	24.5	20
M2.1.2.1	31905.8	80.505	99.7	20
M2.2.2.1	34496.9	79.444	84.0	20
M2.3.2.1	34571.7	72.955	65.2	20
M2.1.2.2	32074.9	48.510	104.9	20
M2.2.2.2	34169.8	51.464	87.2	20
M2.3.2.2	34943.3	51.887	63.1	20

(c) Knapsack problem with $m = 1$ constraint and $n = 80$ variables.

knapsack problem, $m = 1, n = 100$				
version	nodes	time (s)	IPs	solved
BB	297345.3	484.676	0.0	20
BS1	68920.5	128.967	0.0	20
WS	128080.8	224.587	67.0	20
M1.1.1	67369.1	128.665	54.1	20
M1.2.1	67370.1	128.924	40.0	20
M1.3.1	67353.3	128.993	32.9	20
M2.1.1.1	58214.2	198.683	156.9	20
M2.2.1.1	61533.1	179.516	123.0	20
M2.3.1.1	62127.3	177.621	104.6	20
M2.1.1.2	58151.3	112.575	158.1	20
M2.2.1.2	61490.6	118.600	120.1	20
M2.3.1.2	61762.6	118.306	108.7	20
BS2	187306.9	318.524	0.0	20
M1.1.2	73766.2	144.684	54.8	20
M1.2.2	74065.4	144.677	37.9	20
M1.3.2	73865.0	144.306	31.0	20
M2.1.2.1	59512.7	200.754	158.1	20
M2.2.2.1	64489.2	192.211	127.0	20
M2.3.2.1	64330.5	187.803	114.7	20
M2.1.2.2	60470.8	118.479	157.8	20
M2.2.2.2	64943.8	127.428	123.5	20
M2.3.2.2	64711.1	126.525	113.5	20

(d) Knapsack problem with $m = 1$ constraints and $n = 100$ variables.

Table 4.1: Numerical results of the bi-objective, one-dimensional knapsack problems.

knapsack problem, $m = 2, n = 25$				
version	nodes	time (s)	IPs	solved
BB	3932.6	1.3324	0.0	20
BS1	2382.0	0.8426	0.0	20
WS	2049.6	1.0296	10.9	20
M1.1.1	1758.3	0.8999	9.9	20
M1.2.1	1815.6	0.8356	7.0	20
M1.3.1	1851.7	0.8112	5.8	20
M2.1.1.1	1655.3	1.0261	13.3	20
M2.2.1.1	1791.1	0.8898	8.5	20
M2.3.1.1	1831.0	0.8565	7.0	20
M2.1.1.2	1652.7	0.9496	13.3	20
M2.2.1.2	1778.9	0.8690	8.5	20
M2.3.1.2	1830.5	0.8451	6.7	20
BS2	2333.8	0.8314	0.0	20
M1.1.2	1796.0	0.9235	9.6	20
M1.2.2	1821.0	0.8441	6.4	20
M1.3.2	1867.9	0.8253	5.1	20
M2.1.2.1	1668.4	1.0192	13.4	20
M2.2.2.1	1762.0	0.8991	9.3	20
M2.3.2.1	1825.1	0.8711	7.2	20
M2.1.2.2	1664.9	0.9723	13.5	20
M2.2.2.2	1768.1	0.8766	9.1	20
M2.3.2.2	1828.8	0.8598	7.1	20

(a) Knapsack problem with $m = 2$ constraint and $n = 25$ variables.

knapsack problem, $m = 2, n = 50$				
version	nodes	time (s)	IPs	solved
BB	27916.3	18.153	0.0	20
BS1	11788.1	8.339	0.0	20
WS	14270.7	10.507	33.8	20
M1.1.1	10789.7	8.452	26.4	20
M1.2.1	10793.5	8.188	21.2	20
M1.3.1	10795.7	8.116	18.0	20
M2.1.1.1	9888.5	10.873	48.7	20
M2.2.1.1	10140.3	9.437	32.7	20
M2.3.1.1	10521.0	8.774	25.7	20
M2.1.1.2	9840.1	8.396	45.6	20
M2.2.1.2	10130.6	8.422	32.4	20
M2.3.1.2	10401.8	8.288	26.3	20
BS2	16739.8	11.397	0.0	20
M1.1.2	11026.3	8.861	26.1	20
M1.2.2	11024.5	8.860	19.9	20
M1.3.2	11047.4	8.645	16.1	20
M2.1.2.1	10071.8	10.907	45.9	20
M2.2.2.1	10421.4	9.587	31.8	20
M2.3.2.1	10583.2	9.448	24.5	20
M2.1.2.2	9994.1	8.940	46.7	20
M2.2.2.2	10413.4	8.820	32.6	20
M2.3.2.2	10568.9	8.727	25.2	20

(b) Knapsack problem with $m = 2$ constraint and $n = 50$ variables.

knapsack problem, $m = 2, n = 80$				
version	nodes	time (s)	IPs	solved
BB	153938.9	186.330	0.0	20
BS1	36392.0	50.952	0.0	20
WS	58825.7	79.545	54.0	20
M1.1.1	34337.7	50.431	41.7	20
M1.2.1	34333.9	50.312	33.1	20
M1.3.1	34307.1	50.505	26.4	20
M2.1.1.1	31643.7	81.625	100.2	20
M2.2.1.1	32708.9	68.939	76.2	20
M2.3.1.1	32986.3	69.848	63.6	20
M2.1.1.2	31274.5	46.721	102.9	20
M2.2.1.2	32795.7	48.576	76.3	20
M2.3.1.2	33025.8	48.358	63.4	20
BS2	90976.0	116.847	0.0	20
M1.1.2	39745.1	59.321	45.3	20
M1.2.2	40083.2	59.350	31.2	20
M1.3.2	39918.1	58.999	24.5	20
M2.1.2.1	31905.8	80.505	99.7	20
M2.2.2.1	34496.9	79.444	84.0	20
M2.3.2.1	34571.7	72.955	65.2	20
M2.1.2.2	32074.9	48.510	104.9	20
M2.2.2.2	34169.8	51.464	87.2	20
M2.3.2.2	34943.3	51.887	63.1	20

(c) Knapsack problem with $m = 2$ constraint and $n = 80$ variables.

knapsack problem, $m = 2, n = 100$				
version	nodes	time (s)	IPs	solved
BB	428526.3	1074.21	0.0	20
BS1	100962.6	326.98	0.0	20
WS	166108.1	464.71	67.3	20
M1.1.1	98831.5	323.54	54.8	20
M1.2.1	99313.5	325.32	39.0	20
M1.3.1	98770.6	322.65	32.6	20
M2.1.1.1	69951.9	402.48	149.4	20
M2.2.1.1	73433.8	379.33	119.6	20
M2.3.1.1	73424.7	371.63	103.0	20
M2.1.1.2	70172.3	212.40	153.2	20
M2.2.1.2	72824.8	219.73	121.6	20
M2.3.1.2	73651.5	221.96	107.5	20
BS2	271110.8	720.46	0.0	20
M1.1.2	113605.9	381.46	57.5	20
M1.2.2	117188.3	394.57	36.5	20
M1.3.2	113665.3	378.63	29.9	20
M2.1.2.1	70603.0	404.28	150.8	20
M2.2.2.1	77836.0	400.18	121.4	20
M2.3.2.1	76818.2	399.89	110.8	20
M2.1.2.2	72316.9	219.91	148.4	20
M2.2.2.2	78135.2	240.57	121.3	20
M2.3.2.2	77073.0	235.49	112.5	20

(d) Knapsack problem with $m = 2$ constraints and $n = 100$ variables.

Table 4.2: Numerical results of the bi-objective, two-dimensional knapsack problems.

knapsack problem, $m = 3, n = 25$				
version	nodes	time (s)	IPs	solved
BB	5247.4	2.0211	0.0	20
BS1	2628.2	1.0888	0.0	20
WS	2388.6	1.3174	12.0	20
M1.1.1	1970.1	1.1032	9.6	20
M1.2.1	1975.2	1.0055	6.6	20
M1.3.1	1980.4	0.9824	5.4	20
M2.1.1.1	1913.3	1.3484	15.7	20
M2.2.1.1	1935.5	1.0950	8.7	20
M2.3.1.1	1941.4	1.0205	6.5	20
M2.1.1.2	1916.4	1.1913	14.3	20
M2.2.1.2	1936.5	1.0734	8.7	20
M2.3.1.2	1941.7	0.9990	6.6	20
BS2	2617.1	1.0768	0.0	20
M1.1.2	1982.8	1.1242	9.8	20
M1.2.2	1983.1	1.0376	6.8	20
M1.3.2	1975.0	1.0045	5.9	20
M2.1.2.1	1935.4	1.3192	14.4	20
M2.2.2.1	1943.2	1.1729	10.1	20
M2.3.2.1	1935.9	1.1067	8.3	20
M2.1.2.2	1938.9	1.2467	14.3	20
M2.2.2.2	1943.5	1.1135	9.7	20
M2.3.2.2	1934.8	1.0448	7.8	20

(a) Knapsack problem with $m = 3$ constraint and $n = 25$ variables.

knapsack problem, $m = 3, n = 50$				
version	nodes	time (s)	IPs	solved
BB	54430.4	51.5026	0.0	20
BS1	15260.1	17.9276	0.0	20
WS	18112.9	20.5208	36.2	20
M1.1.1	13522.4	16.8431	28.8	20
M1.2.1	13530.7	16.4735	19.0	20
M1.3.1	13576.5	16.4442	15.0	20
M2.1.1.1	12345.3	22.1289	51.2	20
M2.2.1.1	12973.5	19.7592	34.5	20
M2.3.1.1	13014.0	19.6348	28.8	20
M2.1.1.2	12241.1	16.1190	53.6	20
M2.2.1.2	12934.3	16.2933	35.2	20
M2.3.1.2	12908.3	16.1009	30.4	20
BS2	22597.3	24.5736	0.0	20
M1.1.2	14645.7	18.6425	29.6	20
M1.2.2	14573.2	18.0521	16.8	20
M1.3.2	14597.0	17.9793	14.5	20
M2.1.2.1	12617.9	22.3518	54.7	20
M2.2.2.1	13324.9	20.7655	33.4	20
M2.3.2.1	13252.3	20.4366	30.6	20
M2.1.2.2	12682.4	16.8670	56.7	20
M2.2.2.2	13180.2	16.7601	33.6	20
M2.3.2.2	13274.4	16.8497	32.2	20

(b) Knapsack problem with $m = 3$ constraint and $n = 50$ variables.

knapsack problem, $m = 3, n = 80$				
version	nodes	time (s)	IPs	solved
BB	263971.6	724.999	0.0	20
BS1	81609.9	287.899	0.0	20
WS	121360.8	376.247	56.4	20
M1.1.1	80406.5	282.897	47.4	20
M1.2.1	79971.5	279.885	32.2	20
M1.3.1	80089.6	279.686	26.6	20
M2.1.1.1	54187.1	340.389	115.7	20
M2.2.1.1	55915.9	328.411	85.5	20
M2.3.1.1	58486.8	330.347	66.9	20
M2.1.1.2	53396.0	164.755	116.0	20
M2.2.1.2	56572.6	174.131	86.3	20
M2.3.1.2	57452.9	176.657	67.9	20
BS2	140681.4	390.578	0.0	20
M1.1.2	92175.8	334.414	50.5	20
M1.2.2	96339.3	350.148	29.1	20
M1.3.2	96099.5	348.915	24.0	20
M2.1.2.1	54119.5	349.261	112.5	20
M2.2.2.1	59379.8	344.899	89.1	20
M2.3.2.1	60326.6	349.874	74.2	20
M2.1.2.2	54595.6	176.09	119.7	20
M2.2.2.2	62851.9	211.373	88.9	20
M2.3.2.2	61200.8	205.413	76.6	20

(c) Knapsack problem with $m = 3$ constraint and $n = 80$ variables.

Table 4.3: Numerical results of the bi-objective, three-dimensional knapsack problems.

considered subproblems) is increasing, the total computation time decreases. This implies that the reduced computation time to solve IP scalarizations compensates the increase of nodes, which results in a trade-off between the number of explored nodes and the computation time. Another interesting aspect can be observed in M2.α.β.1 and M2.α.β.2. The computation time can be reduced if we do not integrate the augmented weighted Tchebycheff objective level set into the lower bound. This can be explained by the fact that the lower bound improvements of augmented weighted Tchebycheff are only local and do not compensate the computation time needed to integrate the information. The intuitive assumption that the number of explored nodes will then rise significantly is false. So, both our branching strategies work better, if we do not consider the local updates of the lower bound.

We can reach a reduction of the explored nodes by up to 83% (Table 4.2d) and a reduction of the computation time by up to 80% (Table 4.2d) in the best case. The strategies M2.1.1.1 and M2.1.1.2 seem to work best for knapsack problems. In most cases these two strategies have the largest impact on the number of explored nodes. Nevertheless, M2.1.1.2 achieves for all instance sizes the best computation times, since computation time is saved by not integrating the augmented weighted Tchebycheff objective space information into the lower bound. Note that with rising numbers of variables and constraints the hybridization techniques have larger impact on the performance of the branch and bound algorithm.

4.3.2 Bi-objective Assignment Problems

We consider bi-objective assignment problems having $n = \ell^2$ variables,

$$\begin{aligned}
\max \quad & \sum_{i=1}^{\ell}\sum_{t=1}^{\ell} c_{it}^{j}\, x_{it} && j = 1,2 \\
\text{s.t.} \quad & \sum_{i=1}^{\ell} x_{it} = 1 && t = 1,...,\ell \\
& \sum_{t=1}^{\ell} x_{it} = 1 && i = 1,...,\ell \\
& x \in \{0,1\}^{\ell\times\ell}
\end{aligned}$$

with the cost coefficients $c_{it}^{j} \in [50, 100]$. The algorithmic strategy for the solution of IP scalarizations depending on the value of the parameter α is chosen similarly to the previous case of knapsack problems. However, we adapt the boundaries due to the different number of nodes to explore in assignment problems. In M1.1.β, weighted sum scalarizations are solved every 10-th iteration to integer optimality. In M1.2.β, we apply the weighted sum

every 10-th iteration within the first $n \cdot \ell$ iterations. In M1.3.β, we apply the weighted sum every 10-th iteration within the first $n \cdot \ell/3$ iterations, every ℓ-th iteration in the next $n \cdot \ell/3$ iterations and every n-th iteration in the third $n \cdot \ell/3$ iterations. For M.2.α.β.γ, we use the same algorithmic strategy as in hybrid branch and bound for knapsack problems. If a scalarization cannot be applied or an IP with identical objective function has been solved prior, no IP is solved in that iteration.

Due to the total unimodularity of the assignment problem, the weighted sum scalarizations do in general not improve the lower bound sets of subproblems. However, in situations where the weighted sum IP scalarization generates a supported efficient solution, whose corresponding non-dominated point is not an extreme point of the lower bound set, the local upper bounds move closer to the lower bound set. This reduces the gap between upper and lower bound and may lead to a decrease of the explored subproblems. Note that this update of the upper bound set may also have the contrary effect (the number of considered subproblems increases), since it can change the order in which the subproblems are considered. Although in general the weighted sum IP scalarizations are necessary to determine non-dominated points based on which the augmented weighted Tchebycheff scalarization can be applied, they are redundant for assignment problems due to the total unimodularity of their constraint matrix.

Thus, all supported non-dominated extreme points (and their corresponding solutions) are obtained as extreme points of the lower bound set by the linear relaxation of the original problem which is solved in the root node of the branch and bound tree. However, we do not adjust our branch and bound algorithm for totally unimodular problem classes to maintain comparable numerical results and general applicability.

Our branching strategies have a significant impact on the number of explored nodes and the computation time. Again, the local hypervolume gap strategy performs better than the global hypervolume gap strategy. With the local strategy we can reduce the number of explored nodes by up to 39% (Table 4.4c) and the computation time by up to 33% (Table 4.4c). Using the global hypervolume gap strategy we can reduce the number of explored nodes by up to 12% (Table 4.4b) and the computation time by up to 12% (Table 4.4b). We reach a reduction of the explored nodes by up to 46% (Table 4.4d) and a reduction of the computation time by up to 42% (Table 4.4d), in the best case. Again, the strategies M2.1.1.1 and M2.1.1.2 seem to work best for assignment problems in terms of explored nodes. Nevertheless, M2.1.1.2 leads to a better computation time which can be explained by the same argument as before. Furthermore, strategy BS1 works very well and is able to compete with the previously mentioned strategies with respect to number of nodes and computation time without solving a single IP scalarization.

assignment problem, $n = 100$				
version	nodes	time (s)	IPs	solved
BB	3117.0	5.1507	0.0	20
BS1	2422.2	4.1182	0.0	20
WS	3117.0	5.2631	19.2	20
M1.1.1	2425.1	4.1937	14.0	20
M1.2.1	2425.1	4.1564	12.0	20
M1.3.1	2425.1	4.1996	8.4	20
M2.1.1.1	2418.2	4.4063	19.8	20
M2.2.1.1	2420.5	4.2726	14.4	20
M2.3.1.1	2419.5	4.2242	9.7	20
M2.1.1.2	2420.5	4.3474	19.9	20
M2.2.1.2	2420.5	4.2014	14.6	20
M2.3.1.2	2423.0	4.2055	9.8	20
BS2	2948.9	4.9644	0.0	20
M1.1.2	2519.8	4.4349	14.1	20
M1.2.2	2518.7	4.4211	11.1	20
M1.3.2	2516.2	4.4203	7.7	20
M2.1.2.1	2499.7	4.6173	20.4	20
M2.2.2.1	2518.7	4.4661	12.2	20
M2.3.2.1	2516.2	4.4214	8.4	20
M2.1.2.2	2498.1	4.5400	19.6	20
M2.2.2.2	2518.7	4.4451	12.3	20
M2.3.2.2	2516.2	4.3972	8.4	20

(a) Assignment problem with $n = 100$ variables.

assignment problem, $n = 144$				
version	nodes	time (s)	IPs	solved
BB	9661.9	28.2667	0.0	20
BS1	6255.4	18.9362	0.0	20
WS	9610.8	28.1363	33.4	20
M1.1.1	6274.5	18.9323	22.3	20
M1.2.1	6274.5	18.9869	14.3	20
M1.3.1	6274.5	18.9318	9.9	20
M2.1.1.1	6210.9	20.0216	46.0	20
M2.2.1.1	6279.9	19.4670	23.6	20
M2.3.1.1	6271.2	19.1542	12.7	20
M2.1.1.2	6171.0	19.4994	43.6	20
M2.2.1.2	6261.5	19.2851	24.7	20
M2.3.1.2	6270.5	18.8676	12.5	20
BS2	8448.0	24.7742	0.0	20
M1.1.2	6781.8	20.6825	24.1	20
M1.2.2	6779.4	20.5940	13.3	20
M1.3.2	6734.2	20.4270	9.3	20
M2.1.2.1	6472.8	20.8732	44.1	20
M2.2.2.1	6724.2	20.6838	16.8	20
M2.3.2.1	6682.3	20.4212	12.6	20
M2.1.2.2	6500.3	20.2536	42.6	20
M2.2.2.2	6737.7	20.5214	16.8	20
M2.3.2.2	6689.0	20.3987	13.0	20

(b) Assignment problem with $n = 144$ variables.

assignment problem, $n = 225$				
version	nodes	time (s)	IPs	solved
BB	26810.5	160.712	0.0	20
BS1	16142.2	107.909	0.0	20
WS	26810.5	163.666	46.9	20
M1.1.1	16150.0	107.842	32.8	20
M1.2.1	16151.2	109.687	19.7	20
M1.3.1	16164.7	109.073	12.1	20
M2.1.1.1	15424.1	111.181	87.4	20
M2.2.1.1	15964.3	110.004	43.1	20
M2.3.1.1	16112.2	110.372	19.5	20
M2.1.1.2	15554.1	106.748	89.5	20
M2.2.1.2	16008.4	107.641	41.4	20
M2.3.1.2	16106.5	109.735	19.3	20
BS2	24566.2	152.341	0.0	20
M1.1.2	17499.1	117.481	34.2	20
M1.2.2	17591.4	118.804	16.8	20
M1.3.2	17287.8	116.681	11.3	20
M2.1.2.1	16095.7	115.012	85.8	20
M2.2.2.1	17048.8	116.275	31.6	20
M2.3.2.1	17093.3	117.976	17.8	20
M2.1.2.2	16183.0	110.920	79.6	20
M2.2.2.2	17104.7	117.885	32.2	20
M2.3.2.2	17070.0	117.557	16.7	20

(c) Assignment problem with $n = 225$ variables.

assignment problem, $n = 324$				
version	nodes	time (s)	IPs	solved
BB	76643.0	798.644	0.0	20
BS1	47311.6	527.471	0.0	20
WS	75179.2	786.036	61.8	20
M1.1.1	47327.0	530.055	47.8	20
M1.2.1	47332.8	523.562	21.9	20
M1.3.1	47375.6	524.610	15.3	20
M2.1.1.1	40978.0	477.927	157.5	20
M2.2.1.1	43760.5	497.812	82.0	20
M2.3.1.1	44343.9	499.678	52.4	20
M2.1.1.2	41294.4	457.120	159.1	20
M2.2.1.2	43864.6	487.051	81.3	20
M2.3.1.2	44499.7	489.744	52.1	20
BS2	69042.4	723.853	0.0	20
M1.1.2	49367.5	565.719	48.5	20
M1.2.2	49053.0	553.130	23.0	20
M1.3.2	49221.4	554.594	15.2	20
M2.1.2.1	41840.5	489.647	150.2	20
M2.2.2.1	45621.0	520.469	67.2	20
M2.3.2.1	46915.9	533.692	39.4	20
M2.1.2.2	42726.6	474.220	156.2	20
M2.2.2.2	45901.6	513.111	67.1	20
M2.3.2.2	46902.8	526.440	43.5	20

(d) Assignment problem with $n = 324$ variables.

Table 4.4: Numerical results of the bi-objective assignment problems.

4.3.3 Bi-objective Discrete Uncapacitated Facility Location Problems

We consider discrete uncapacitated facility location problems of the form

$$\begin{aligned}
\min \quad & \sum_{i=1}^{\ell}\sum_{t=1}^{q} c_{it}^{j}\, x_{it} + \sum_{t=1}^{q} \kappa_t^j \zeta_t && j = 1,2 \\
\text{s.t.} \quad & \sum_{t=1}^{q} x_{it} = 1 && i = 1,...,\ell \\
& x_{it} \leq \zeta_t && i = 1,...,\ell,\ t = 1,...,q \\
& x \in \{0,1\}^{\ell \times q} \\
& \zeta \in \{0,1\}^{q}
\end{aligned}$$

where ℓ is the number of customers and q the number of facilities. We randomly generate coordinates of ℓ customers and q facilities in a square with length 200. The costs of the first objective function correspond to the L_1-distances between the customers and facilities, while the costs of the second objective function are randomly generated (i.e., $c_{it}^2 \in [1, 200]$) and $\kappa_t^j \in [200, 400]$. The number of variables is $n = (\ell + 1)\, q$. We restrict the numerical tests to problems where the number of facilities is 20% of the number of customers. Again, we need to specify when and how often integer scalarizations are applied: We use the same methods as before but adapt the boundaries due to the different number of nodes to explore in discrete facility location problems. In M1.1.β, we apply the weighted sum IP scalarization every 10-th iteration. In M1.2.β, we apply the weighted sum every 10-th iteration within the first $n^2/4$ iterations. In M1.3.β, we apply the weighted sum scalarization every 10-th iteration in the first $n^2/4$ iterations, every $n/2$-th iteration in the next $n^2/4$ iterations and every n-th iteration in the third $n^2/4$ iterations. In M.2.α.β.γ, we operate analogous to the methods used for knapsack and assignment problems. If a scalarization cannot be applied or an IP with identical objective function has been solved prior, no IP is solved in that iteration.

Again, both new branching strategies have an impact on the number of explored nodes and the computation time. The local strategy, once more, leads to better results, namely a reduction of the explored nodes by up to 52% (Table 4.5d) and a reduction of the computation time by up to 45% (Table 4.5d). With the global hypervolume gap strategy we can reach a reduction of the explored nodes by up to 24% (Table 4.5d) and a reduction of the computation time by up to 21% (Table 4.5d). In the best case, we can reach a reduction of the explored nodes by up to 57% (Table 4.5d) and of the computation time by up to 50% (Table 4.5d). Once again, M2.1.1.2 seems to be the best choice with respect to the number of explored nodes and with a rising number of variables it is also the best

facility location problem, $n = 48$				
version	nodes	time (s)	IPs	solved
BB	1431.1	1.0851	0.0	20
BS1	999.8	0.7640	0.0	20
WS	1431.1	1.4550	15.4	20
M1.1.1	1000.2	0.8354	11.0	20
M1.2.1	1000.2	0.8219	8.8	20
M1.3.1	1000.2	0.8243	7.0	20
M2.1.1.1	999.7	0.9361	13.2	20
M2.2.1.1	1000.2	0.8361	9.9	20
M2.3.1.1	1000.2	0.8079	7.9	20
M2.1.1.2	999.3	0.8536	12.8	20
M2.2.1.2	1000.2	0.8182	9.9	20
M2.3.1.2	1000.2	0.8037	7.9	20
BS2	1268.6	0.9714	0.0	20
M1.1.2	1041.8	0.8654	12.3	20
M1.2.2	1041.8	0.8527	9.4	20
M1.3.2	1041.8	0.8592	7.4	20
M2.1.2.1	1036.5	0.9722	15.7	20
M2.2.2.1	1041.8	0.8761	10.3	20
M2.3.2.1	1041.8	0.8735	7.6	20
M2.1.2.2	1034.6	0.9678	15.1	20
M2.2.2.2	1041.8	0.8843	10.4	20
M2.3.2.2	1041.8	0.8654	7.6	20

(a) Facility location problem with 15 customers and 3 facilities

facility location problem, $n = 84$				
version	nodes	time (s)	IPs	solved
BB	7949.1	12.6393	0.0	20
BS1	4626.7	7.9303	0.0	20
WS	7490.2	12.2058	35.0	20
M1.1.1	4627.2	8.1249	26.5	20
M1.2.1	4627.2	8.1149	18.2	20
M1.3.1	4626.4	8.0610	13.9	20
M2.1.1.1	4527.8	9.2394	49.0	20
M2.2.1.1	4601.6	8.4311	26.5	20
M2.3.1.1	4610.4	8.2569	18.9	20
M2.1.1.2	4526.1	8.4415	45.3	20
M2.2.1.2	4591.1	8.1289	24.3	20
M2.3.1.2	4605.2	8.1312	19.8	20
BS2	7084.4	11.5822	0.0	20
M1.1.2	4873.8	8.7719	26.4	20
M1.2.2	4874.8	8.7534	17.5	20
M1.3.2	4894.1	8.7031	12.6	20
M2.1.2.1	4673.6	9.5755	49.7	20
M2.2.2.1	4832.0	8.9275	23.6	20
M2.3.2.1	4812.8	8.8050	17.4	20
M2.1.2.2	4628.9	8.7019	46.2	20
M2.2.2.2	4825.9	8.7491	24.4	20
M2.3.2.2	4807.5	8.5984	17.5	20

(b) Facility location problem with 20 customers and 4 facilities

facility location problem, $n = 130$				
version	nodes	time (s)	IPs	solved
BB	17461.9	51.6307	0.0	20
BS1	10684.0	34.5157	0.0	20
WS	16795.9	50.9317	51.4	20
M1.1.1	10753.9	35.4356	37.1	20
M1.2.1	10753.9	35.2764	25.5	20
M1.3.1	10753.9	35.1007	19.2	20
M2.1.1.1	10104.4	38.3909	89.7	20
M2.2.1.1	10678.1	35.7434	39.4	20
M2.3.1.1	10722.3	35.6262	27.7	20
M2.1.1.2	10103.0	34.5003	86.0	20
M2.2.1.2	10691.2	35.3730	39.1	20
M2.3.1.2	10718.6	35.1690	27.5	20
BS2	15474.7	46.5130	0.0	20
M1.1.2	11548.8	38.4891	39.3	20
M1.2.2	11601.4	38.5848	24.6	20
M1.3.2	11535.1	38.2917	18.0	20
M2.1.2.1	10684.3	39.8639	81.5	20
M2.2.2.1	11381.8	39.1988	37.2	20
M2.3.2.1	11336.0	38.7754	28.5	20
M2.1.2.2	10695.5	36.9098	78.3	20
M2.2.2.2	11341.6	38.2646	39.6	20
M2.3.2.2	11352.1	38.0063	25.9	20

(c) Facility location problem with 25 customers and 5 facilities

facility location problem, $n = 186$				
version	nodes	time (s)	IPs	solved
BB	67369.3	373.238	0.0	20
BS1	31844.1	203.145	0.0	20
WS	62192.8	349.741	70.0	20
M1.1.1	32106.6	206.244	53.1	20
M1.2.1	32097.9	206.851	33.4	20
M1.3.1	32148.5	207.199	23.8	20
M2.1.1.1	28384.2	224.486	186.8	20
M2.2.1.1	31074.5	218.314	101.2	20
M2.3.1.1	32004.8	209.608	50.1	20
M2.1.1.2	28558.8	186.010	172.7	20
M2.2.1.2	30946.4	202.329	97.8	20
M2.3.1.2	32011.8	207.715	47.5	20
BS2	50759.0	292.344	0.0	20
M1.1.2	35150.1	230.687	55.5	20
M1.2.2	35789.8	233.967	30.8	20
M1.3.2	35255.0	233.163	22.0	20
M2.1.2.1	29704.3	230.016	172.7	20
M2.2.2.1	33959.7	236.351	81.8	20
M2.3.2.1	34228.0	233.553	50.2	20
M2.1.2.2	30412.6	202.719	167.4	20
M2.2.2.2	34511.8	229.970	70.0	20
M2.3.2.2	34405.0	229.021	47.1	20

(d) Facility location problem with 30 customers and 6 facilities

Table 4.5: Numerical results of the bi-objective integer facility location problems

choice regarding the computation time. With a smaller number of variables, BS1 leads to good results with respect to both aspects without solving a single IP.

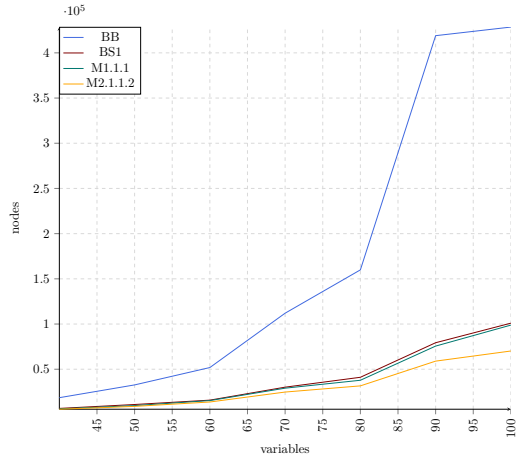
4.3.4 Summary

In all of the three tested problem classes (knapsack, assignment, discrete uncapacitated facility location) a significant reduction of the number of explored nodes and the computation time can be realized with all presented combinations of the hybrid branch and bound approach. With increasing problem size (number of variables) the impact of the presented augmentations increases. Furthermore, the approaches perform better on problems where the gap between Y_N and the solution of the linear relaxation is larger compared to *totally unimodular* problems. The reduction in terms of the number of branch and bound nodes and runtime we achieve with the proposed methods as compared to plain branch and bound is visualized in Figure 4.4 for varying instance sizes.

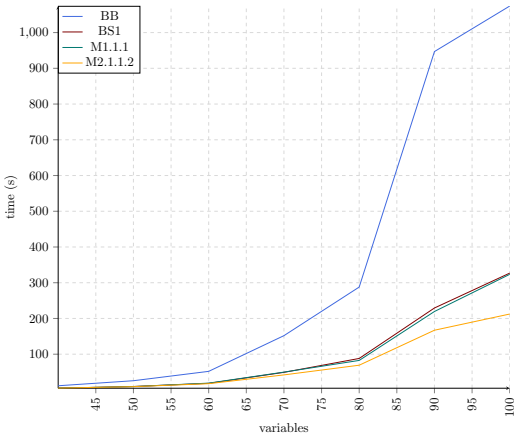
The local hypervolume gap strategy for the node selection outperforms the global hypervolume gap strategy in our numerical tests. A reason for this is that in the global hypervolume gap strategy many small search zones can add up to a large gap although the lower bound might be quite close to the non-dominated points. The local hypervolume gap strategy chooses the node with the largest search zone, which has the biggest potential to reduce this gap. Moreover, the local hypervolume gap strategy aims at an uniform distribution of points in the incumbent list. In our numerical test, M2.1.1.2 turns out to be the best choice in most cases with respect to the number of explored nodes and computation time. In this version, we use the local hypervolume gap strategy for the choice of the active node, every 10-th iteration the weighted sum IP scalarization is applied and every 50-th iteration we apply the augmented weighted Tchebycheff scalarization instead. Furthermore, the objective space information gained by the augmented weighted Tchebycheff scalarization is not used to update the lower bound set, since its local improvements do not compensate the increased computation time. Although we need to solve more IPs than in most other approaches, the computation time is the lowest compared to the others. So, using the augmented weighted Tchebycheff scalarization in the beginning of the branch and bound works best. Due to the likelihood of finding unsupported non-dominated points in the early stages of the algorithm, the upper bound can be further improved. This results in a higher probability of fathoming a node by dominance. Nevertheless, with version BS1 we also achieve a remarkable reduction in terms of the number of explored nodes and computation time by using the local hypervolume gap strategy for node selection.

Concluding, we propose two approaches to incorporate objective space information in bi-objective branch bound algorithms. By using the local or global (approximated) hy-

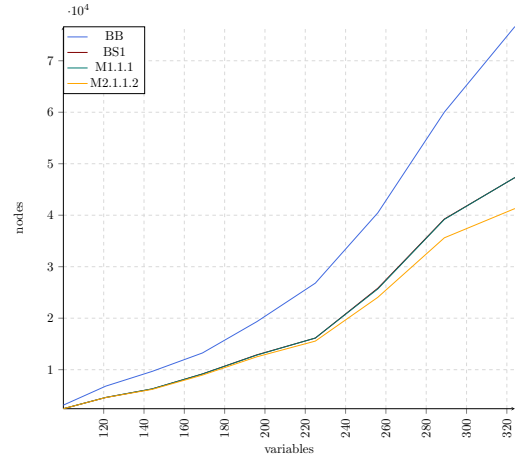
pervolume gap as node selection criterion, we guide the search of the branch and bound algorithm in promising directions. Additionally, we adaptively solve scalarizations to integer optimality to improve the lower and upper bound set. The numerical results show the effectiveness of both approaches and, in particular, of their combination. The dynamic branching rule based on the local (approximated) hypervolume gap has large impact on the number of explored subproblems, is compuationally efficient and can be easily integrated in other multi-objective branch and bound algorithms. While we evaluate in this chapter the individual contributions of our augmentations on a generic bi-objective branch and bound, we will continue to extend our ideas in the following chapter.



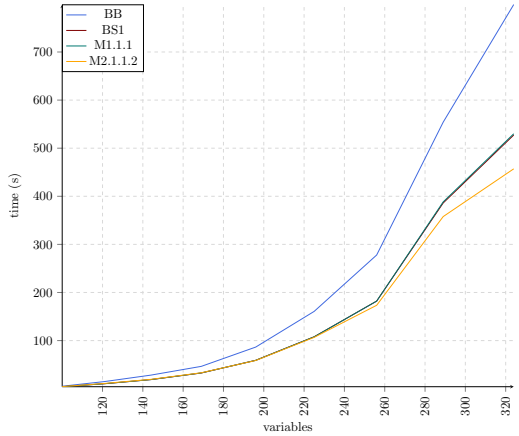
(a) Number of nodes for 2-D knapsack problems.



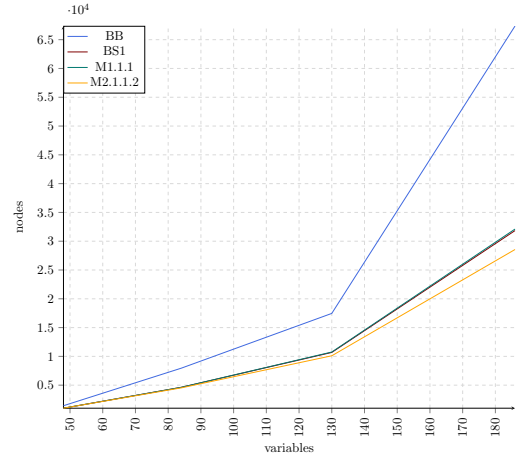
(b) Runtime for 2-D knapsack problems.



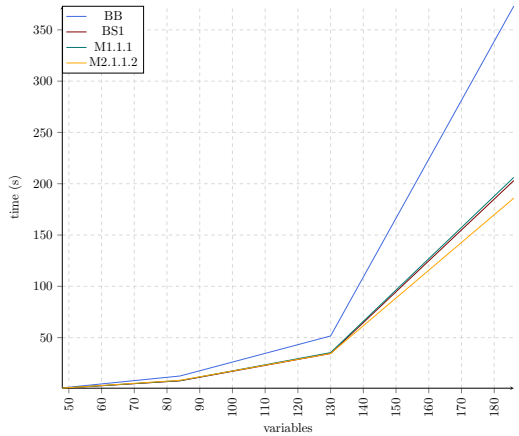
(c) Number of nodes for assignment problems.



(d) Runtime for assignment problems.



(e) Number of nodes for facility location problems.



(f) Runtime for facility location problems.

Figure 4.4: A visualization of the node and runtime reduction of the considered instances.

5 Adaptive Improvements of Multi-objective Branch and Bound

In the previous chapter, it is shown that the incorporation of objective space information can significantly improve bi-objective branch and bound algorithms. This observation motivates us to extend these approaches to the multi-objective case with $p \geq 3$ objectives. In higher dimension, however, the non-dominated set is in general not naturally ordered anymore. Moreover, the geometrical structure in the objective space is more complex. Thus, the use of objective space information has to be restructured for the multi-objective case.

We adapt the most promising approaches of Chapter 4 in combination with further modifications to improve the performance of branch and bound for MOILPs with $p \geq 3$ objectives. More precisely, we show how objective space information can improve the computational performance of multi-objective branch and bound. Thereby, we focus on dynamic branching strategies and the usage of IP scalarizations.

As already mentioned, there are two main shortcomings of multi-objective branch and bound algorithms. Firstly, the bounding is weaker, compared to its single-objective counterpart and secondly, optimized single-objective solvers lead to the supremacy of objective space methods. Therefore, we utilize those single-objective solvers to solve scalarized integer programs and use the obtained information to possibly improve the lower and upper bound set. Furthermore, we present dynamic node selection strategies. Although it is common to use dynamic strategies in the single-objective case, they are rarely applied in multi-objective branch and bound.

Contribution Most of this chapter has been already published in Bauß et al. (2023).

Organization of the Chapter The remaining of the chapter is organized as follows. In Section 5.1, we propose two different new multi-objective node selection strategies to consider the most promising subproblems first. In Section 5.2, methods to improve the bound sets by using scalarization techniques to benefit from objective space information are presented. Thereby, we propose, among other things, a warmstarting technique for the bound sets and a partial enumeration approach. The numerical tests of these approaches are presented in Section 5.3 to show their impact.

5.1 A New Multi-objective Node Selection Strategy

The order in which nodes in a branch and bound tree are explored has a significant impact on the total number of created nodes and therefore on the computation time. Choosing the right nodes and obtaining good feasible solutions in early stages of the algorithm can lead to a decrease of nodes to explore, since they could dominate numerous other nodes, which as a result can be fathomed. The arising difficulty is to use a good node selection strategy that causes a desirable reduction of nodes. Therefore, instead of using a static strategy, like, e.g., the depth-first strategy, we propose a new dynamic strategy.

The underlying principle of this strategy is a direct extension of a node selection strategy that is frequently used in the single-objective case. There, in each iteration the node with the largest gap between upper and lower bound is selected (see, e.g., Dechter and Pearl, 1985; Morrison et al., 2016). In the multi-objective case, there are numerous approaches to measure the gap between the lower bound set and the set of local upper bounds and/or the points in the incumbent list. See Chapter 6, for a comparison of different gap measures and the according queuing of subproblems. We propose to use the so-called approximate hypervolume gap (cf. Section 4.1) to select the active subproblem from the queue, where we rely on the definition of the hypervolume by Zitzler and Thiele (1999). Like in its single-objective counterpart, in every iteration the node with the largest gap is selected. Thereby, we distinguish two different ways to approximate the hypervolume gap.

The first approach is called the local hypervolume gap (see Section 4.1 for the bi-objective case). Instead of measuring the total gap between the lower and upper bound set, we only consider the largest gap between a single local upper bound and the lower bound set, i.e., we only consider the volume of the largest search zone. Again, we refer to Klamroth et al. (2015) for detailed analysis of search zones, search regions, corresponding local upper bounds and their defining non-dominated points.

Figures 5.1(a) and 5.1(b) illustrate the local hypervolume gap approach for a bi-objective example. The points $z^1, \dots, z^4 \in \mathcal{U}$ are points of the upper bound set $\mathcal{U}$. Furthermore, the points $lu^1, \dots, lu^3$ are their corresponding local upper bounds. Note that we only need to consider the local upper bounds that are located above the lower bound set $\mathcal{L}$ of the active node ν. The lower bound set $\mathcal{L}$ is illustrated by the blue line and is obtained by solving the linear relaxation of the corresponding subproblem in node ν. For every local upper bound the approximated volume of the corresponding search zone is computed as the hypervolume of the simplex spanned by a local upper bound and p spanning points on the lower bound set. Those spanning points w.r.t. a local upper bound lu are defined as its axis-parallel projections on the lower bound set, i.e., $sp^j(lu) \coloneqq \{l \in \mathcal{L} \colon l_{p+1-j} = lu_{p+1-j}\}$ for $j = 1, \dots, p$. Thus, the hypervolume gap between a local upper bound lu and the lower

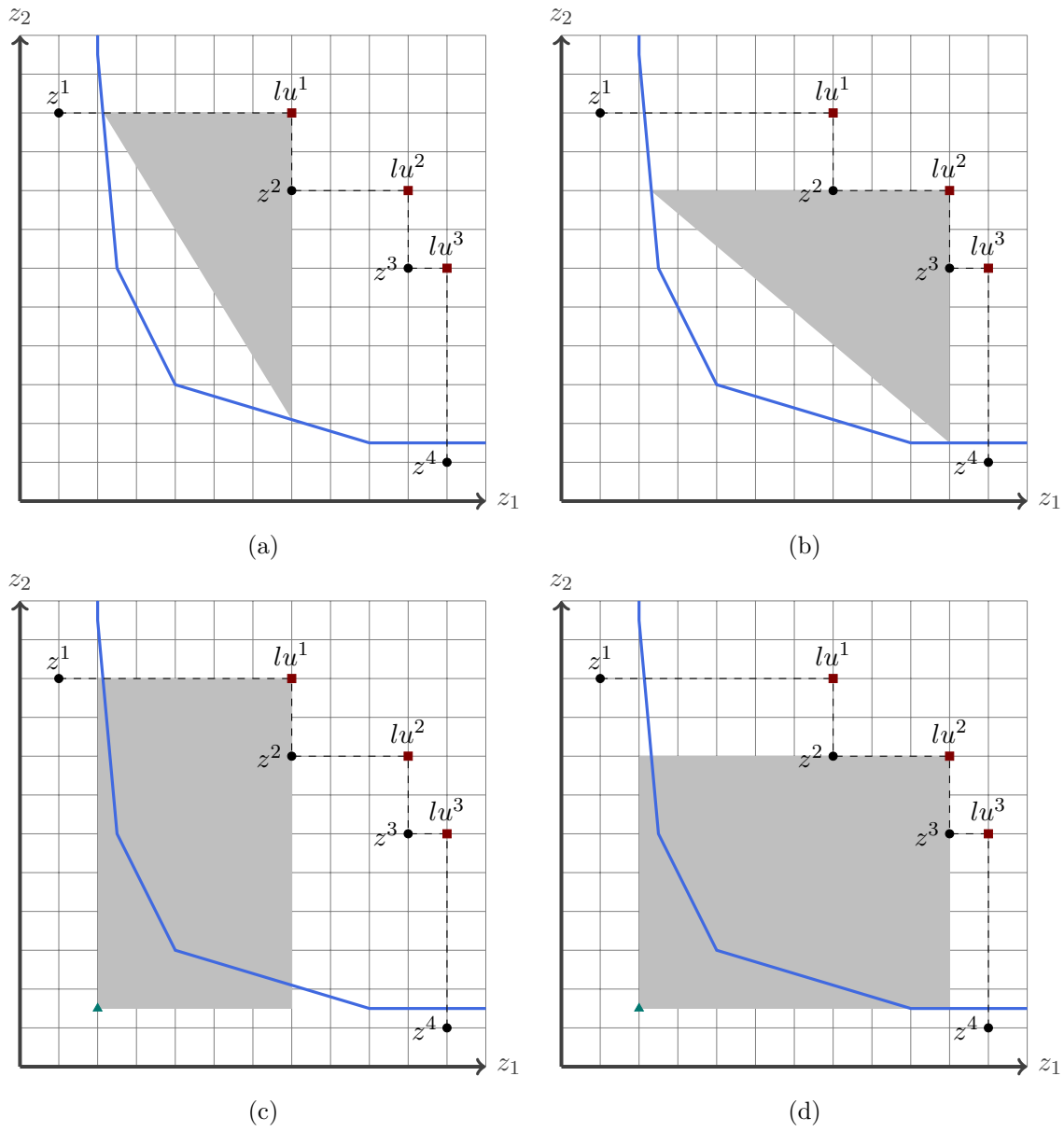


Figure 5.1: A bi-objective example of computation of the two different approximated hypervolume gap approaches. In (a) and (b), the the approximated hypervolume gap (gray) is visualized for the local upper bounds lu^1 and lu^2. In (c) and (d), the hypervolume of the box (gray) defined by the local ideal point and the local upper bound lu^1 respectively lu^2 is shown.

bound set $\mathcal{L}$ is given by

$$hg(lu) \coloneqq \frac{|\det(G)|}{p!},$$

with $G := (sp^1 - lu, \ldots, sp^p - lu) \in \mathbb{R}^{p \times p}$.

Let $K \subset \mathcal{D}(\mathcal{U})$ be the subset of all local upper bounds which are located above the lower bound set of node node ν. Then, the local hypervolume gap of node ν is defined as the largest hypervolume spanned by a local upper bound in K and the corresponding spanning points, i.e.,

$$lhg(\nu) := \max_{i=1,\ldots|K|} hg(lu^i).$$

Obviously, $lhg(\nu)$ is in general only a rough approximation since the real hypervolume of the search zone is underestimated by neglecting possibly large parts. Even though this approximation simplifies the computation significantly compared to the computation of the real hypervolume of a search zone, it gets too time consuming with an increasing number of objective functions. Both, the number of local upper bounds and the number of facets of the lower bound set, increase substantially with the number of objectives. Since the projection of the local upper bounds on the facets of the lower bound set to determine the spanning points requires a significant amount of the total computation time, we simplify the computation at this point further. Note that this is the direct extension of the bi-objective local hypervolume gap presented in Section 4.1.

The second approach to measure the gap between the lower bound $\mathcal{L}$ and the upper bound $\mathcal{U}$ is to compute the *hypervolume of a search zone box* that is defined by a local upper bound lu and the local ideal point of the lower bound set l^I, defined by $l^I_j := \min_{l \in \mathcal{L}} l_j, j \in \{1, \ldots, p\}$. Therefore, the hypervolume of the box defined by local upper bound lu is given by

$$hb(lu) := \prod_{j=1}^{p} (lu_j - l^I_j).$$

Again, this is computed for every local upper bound located above the lower bound and afterwards the volume of the largest box is assigned to the corresponding node. Consequently, the gap between the lower bound and upper bound in node ν, using the hypervolume of a search zone approach, is given by

$$hsz(\nu) := \max_{i=1,\ldots,|K|} hb(lu^i).$$

When new nodes are created by branching, the approximated hypervolume gap of the parent node is assigned to the child nodes to avoid the computation of the lower bound set before the child node becomes active. Note that the set of local upper bounds is initialized with the point $(M, \ldots, M)^\top \in \mathbb{R}^p$ with a sufficiently large value $M \gg 0$. This allows us to immediately apply this node selection strategy at the beginning of our algorithm.

In Figures 5.1(c) and 5.1(d), the hypervolume of a search zone box approach is illus-

trated. The gray area indicates the hypervolume of the box spanned by a local upper bound and the local ideal point of the lower bound, which is illustrated by the green triangle.

In Algorithm 6, the node with the largest assigned hypervolume gap is selected in Step 1. The value of the hypervolume gap is updated in Step 4 if the node cannot be fathomed.

5.2 Solving IP Scalarizations to Improve the Upper and Lower Bound Set

In this section, we propose ways to integrate objective space methods into a multi-objective branch and bound framework. By solving suitable scalarizations to integer optimality, we obtain non-dominated points and thus objective space information that can be used to improve the lower as well as the upper bound set. Let $\bar{x}$ be the integer optimal solution obtained by solving a scalarization of the underlying problem. Since $z(\bar{x})$ is non-dominated, $\bar{x}$ can be added to the incumbent list (if not contained already) and $z(\bar{x})$ can be added to $\mathcal{U}$, which improves the upper bound set. Additionally, depending on the used scalarization technique, the lower bound set might be improved. An improved lower bound reduces the area where possibly new non-dominated points could be found and an improved upper bound set leads to a higher fathoming rate.

5.2.1 Warmstarting the Bound Sets

A branch and bound algorithm benefits from good bound sets and the earlier good bounds are obtained the more impact it has on the performance. Therefore, we present a warmstarting approach for the bound sets. Recall, every optimal solution of (WS_λ) is at least weakly efficient for $\lambda \in \Lambda_0$ but efficient for $\lambda \in \Lambda$, regarding a (MO01LP). This scalarization can only determine supported non-dominated points (cf. Section 3.1).

We solve a limited number of weighted sum scalarizations with different weights $\lambda \in \Lambda$ in a preprocessing step of the branch and bound algorithm. This produces a warmstart of the lower and upper bound set. For the bi-objective case, this idea is proposed in Bökler et al. (2023). The authors use an outer approximation algorithm to generate $\mathrm{conv}(Y)_N$, that can be used as an initial lower bound set in a multi-objective branch and bound algorithm. However, this approach is way more difficult for $p \geq 3$. It has to deal with similar difficulties as the dichotomic search scheme approach (see Aneja and Nair, 1979; Przybylski et al., 2010a, 2019). Hence, we use a predefined weight set $\bar{\Lambda}$ to overcome these problems. For every $\lambda \in \bar{\Lambda}$, a weighted sum scalarization is solved in the root node of the branch and bound tree.

For each $\lambda \in \Lambda$ the scalarized problem (WS_λ) can be solved with a single-objective integer linear programming solver. Let $\bar{x}$ be the optimal solution of (WS_λ), then $z(\bar{x})$ is a (supported) non-dominated point. Hence, we can add this solution to the incumbent list, if it is not contained already, and update the upper bound set and the list of corresponding local upper bounds. Additionally, we obtain further objective space information, that can possibly improve the lower bound set of all nodes that are explored during the algorithm. The optimal solution yields a level set $\{y \in \mathbb{R}^p \colon \lambda^\top y = \lambda^\top z(\bar{x})\}$, which implies the valid inequality $\lambda^\top z(x) \geq \lambda^\top z(\bar{x})$ for all $x \in X$. Since this inequality is obtained by solving a scalarization of the root node problem, it holds for every subproblem. In Chapter 4, the level sets were integrated in an already computed lower bound set by computing the potential cuts of a lower bound set $\mathcal{L}$ and the level set. However, if more than two objectives are considered this would require a lot more computational time, because of the large amount of geometrical operations. Therefore, we avoid this problem by adding the corresponding inequality to the integer programming formulation of every subproblem. During the algorithm the lower bounds are improving which might cause redundancy with respect to these inequalities. Hence, in each iteration we check for redundancy in the active node and delete redundant inequalities.

In Algorithm 6, the warmstart of the lower bound set is integrated in Step 0.

5.2.2 Improving the Upper Bound Set by ε-constraint Scalarizations

Since the weighted sum approach, which is applied as a warmstarting technique for the lower bound set and the incument list, only generates supported non-dominated points, it might be useful to also use other scalarization techniques, that can compute unsupported non-dominated points. In Section 4.2, the augmented weighted Tchebycheff scalarization is used to improve the upper bound by obtaining possibly unsupported non-dominated points. Additionally, the corresponding level set is used to improve the lower bound set. Unfortunately, the updated lower bound set had nearly no impact on the performance, since the lower bound improves just locally and is computationally hard to handle due to its (in general) non-convex structure.

Since we are not updating the lower bound set in this step and only aim at computing unsupported non-dominated points we rely on the ε-constraint scalarization, which we introduced in Section 3.1.1. Recall, every optimal solution of (ε^k-C) is weakly efficient. If additionally the optimal solution of (ε^k-C) is unique, it is an efficient solution of the multi-objective problem. Furthermore, all efficient solutions are optimal solutions of (ε^k-C) for some vector $\varepsilon \in \mathbb{R}^p$, i.e., all efficient solutions can be determined using the ε-constraint scalarization (cf. Section 3.1.1). We apply the ε-constraint method proposed in Kirlik

and Sayın (2014) that guarantees the efficiency of the generated solutions by using the lexicographic optimization approach.

We adaptively solve ε-constraint scalarizations of the root node problem to integer optimality to obtain possibly unsupported non-dominated points. The used ε is obtained by the local upper bound lu, which spans the local hypervolume gap or the hypervolume of a search zone box of the corresponding node. Since we consider integer programs with integer coefficients, ε can be chosen as $\varepsilon := (lu_1 - 1, \dots, lu_p - 1)^\top$. In the best case, the obtained solution maps to a non-dominated point which has not been found before and can thus be added to the incumbent list. However, there is no guarantee that this point is an unsupported non-dominated point. It is not even guaranteed that the scalarized problem is feasible. Nevertheless, if a new non-dominated point is found the upper bound set is improved, which improves the fathoming rate.

The ε-constraint scalarization is applied between Step 2 and Step 3 of Algorithm 6.

5.2.3 Using Simple Lower Bound Sets

The methods proposed so far considered only scalarizations of the root node problem, such that the obtained objective space information can be integrated into every node and the corresponding subproblem, respectively. However, it is also possible to use scalarizations in subproblems of the branch and bound. Then, the obtained information does not hold for every node, but for all child nodes in the corresponding branch. Obtained solutions are efficient for the subproblems but in general not efficient for the underlying problem, since they might be dominated by solutions in other branches.

So, solving IP scalarizations in subproblems can be very time consuming and the obtained information might even be useless. However, solving a weighted sum scalarization to integer optimality compensates in some situations the additional costs. Instead of computing the complete lower bound set which might have many extreme supported points and facets, we adaptively solve a single weighted sum scalarization to integer optimality. As already discussed, the level set of a scalarization in an optimal solution is a valid lower bound on the objective values of the feasible solutions of the subproblem. So, we save the time of computing the lower bound and use only the level set, obtained by solving the weighted sum scalarization, as the lower bound set. Of course, this bound is weaker in most of the covered region compared to the lower bound obtained by solving the linear relaxation. This can be partially compensated by adding the so-called *extreme facets* of the parent node to the lower bound set. By extreme facets we denote the facets of a lower bound set that are parallel to the axes. Thus, the simple lower bound set consists of $p+1$ facets. Then, the obtained inequality holds for every child node of this branch and can

therefore be added as a constraint to the corresponding subproblems. Note that the test for redundancy of those inequalities is done in the same way as described in Section 5.2.1. In the best case, the obtained solution is efficient and has not been found before. Then, it can be added to the incumbent list and the upper bound is improved.

A crucial component in this approach is the choice of the weight $\lambda \in \Lambda$, which should be selected depending on the properties of the corresponding active node. Therefore, we take into consideration all local upper bounds that were still located above the lower bound set in the parent node. Then, for each objective k, we choose among the considered local upper bounds the one with minimal objective value lu_k. Those p points define a hyperplane $\mathcal{H}$. The dichotomic search approach uses the normal vector $\mu \in \mathbb{R}^p$ of $\mathcal{H}$ as the weight λ. Unfortunately, for $p \geq 3$ this normal vector μ may have negative components and cannot be used as weighting vector. Nevertheless, if μ is componentwise non-negative, we use it as weighting vector $\lambda = \mu$. Otherwise, we use the weight $\lambda = (1/p, \ldots, 1/p)^\top \in \mathbb{R}^p$.

5.2.4 Algorithmic Control of the Presented Approaches

In the previous sections, improved components of multi-objective branch and bound algorithms are proposed. However, their algorithmic control is not covered yet, in particular, it is not specified when and how often IP scalarizations should be solved. Since the costs of solving a scalarization to integer optimality are relatively high, this seems to be an important decision. In the first place, we want to obtain as much objective space information as possible. As a result, the lower and upper bound will be improved significantly. These improved bounds increase the probability of fathoming by dominance and reduce the size of the search region. Both aspects will reduce the number of explored nodes, which will in turn reduce the total computational time. However, solving an excessive number of IP scalarizations will increase the computation time. In addition, if IP scalarizations are applied too often, there is an increasing chance that solved scalarizations do not provide new objective space information and are therefore redundant.

Obviously, there exists a trade-off between the decrease of the created nodes and the decrease of the total computation time. It is therefore necessary to find proper conditions, when to solve scalarizations to integer optimality. It is promising to already have good bounds in the early stages of the algorithm. This would lead to an improved fathoming rate from the beginning. Thus, we use a warmstarting approach, which solves IP scalarizations in a preprocessing step. Like mentioned in Section 5.2.1, we use a predefined weight set for that. Preliminary numerical tests have shown, that already a small number of scalarizations solved to integer optimality have a high impact on the performance. We therefore use a predefined weight set $\bar{\Lambda}$ with $|\bar{\Lambda}| = p + 1$. As weight vectors $\lambda \in \bar{\Lambda}$ we

use the standard unit vectors and additionally a weight vector with equal weights, i.e., $\lambda = (1/p, \dots, 1/p)^\top \in \mathbb{R}^p$. Note that we add a small augmentation term to the standard unit vectors to guarantee efficiency.

Similar observations can be made, when the ε-constraint method is used to improve the upper bound. Preliminary tests have shown, that it is more promising to solve these IP scalarizations in the early stages of the algorithm. This increases chances of obtaining unsupported non-dominated points early, which improves the upper bound set. Obviously, when the ε-constraint scalarization is applied too often, we probably waste a lot of time by solving multiple problems to integer optimality without any benefit.

By using the simple lower bound approach instead of computing the complete lower bound set, some information is lost. The simple lower bound is in general weaker and there is no information about extreme points of the lower bound set. In the numerical tests, presented in Section 5.3, we will use the *most often fractional rule* (cf. Section 3.2.2). This rule cannot be applied when the simple lower bound is used, as it is based on a single integer solution and thus there is no fractional variable. However, in this case we use the *sum of ratios* branching rule that is presented in Bazgan et al. (2009) for multi-objective knapsack problems. A detailed description of this branching rule is given in Section 6.1. This can be considered as a direct extension of the basic single-objective branching rule for knapsack problems (cf. Kellerer et al., 2004). Note that this rule can also be applied to other problem classes like facility location problems or generalized assignment problems by interpreting the facility opening costs respectively the workload as weight.

We decided to apply the simple lower bound approach at certain levels of the branch and bound tree. This means, that we compute the simple lower bound instead of the complete bound set, when there is a certain number of fixed variables in the active node. Since there is in general a high number of nodes at deeper levels in the tree, a high amount of problems is solved to integer feasibility, resulting in possibly rising computation times. The efficiency and impact of the presented approaches are presented in the next section.

5.3 Numerical Tests

All presented algorithms were implemented in Julia 1.9.0 and the linear relaxations (for the lower bound set) were solved with Bensolve 2.1 (Löhne and Weißing, 2017). The scalarizations were solved to integer optimality with CPLEX 20.1. The numerical test runs were executed on a single core of a 3.20 GHz Intel® Core™ i7-8700 CPU with 32 GB RAM. Note that the implementation of the proposed algorithms is publicly available (Bauß and Stiglmayr, 2023a).

We present different combinations of our presented approaches and compare their per-

formance to the basic multi-objective branch and bound algorithm, which serves a baseline implementation. The results are evaluated regarding the average number of explored nodes and the average computational time over 10 instances. The time limit on solving a single instance is set to two hours.

Additionally to the basic branch and bound approach, presented in Section 3.2.2, we evaluate different combinations of the proposed approaches. These approaches show measurable impact in our test runs on different sets of problems. All considered branch and bound configurations are described in the following.

- **BB**. The basic branch and bound.
- **NS(.)**. The basic branch and bound, but with the dynamic node selection strategy, presented in Section 5.1. We distinguish between NS(LHG), when the local hypervolume gap is used, and NS(HSZ), when the hypervolume of the search zone box is considered.
- **WST**. Same procedure as NS(.), but using a warmstart of the bound sets.
- **EC**. Same procedure as WST. Additionally, ε-constraint scalarizations are solved. The scalarization is applied every n-th iteration within the first $p\,n^2$ iterations.
- **SLB**. Same procedure as EC, but every fifth level of the branch and bound tree, the simple lower bound is considered, instead of solving the linear relaxation.
- **+TE**. For every problem class, we consider the best performing approaches regarding the number of nodes respectively the total time. In those approaches if only 10 or less variables are free, we enumerate all $2^{10} = 1024$ solutions.

Note that the chosen parameters yield from preliminary numerical experiments on a different set of instances. Of course, they are not optimized and we do not change these parameter values for different problem classes, since we aim to show the impact of these approaches on a variety of problems. Finally, we present the considered problem classes and benchmark instances:

i) Knapsack problems (KP) benchmark instances from Kirlik and Sayın (2014). The instances with 3 objectives and 40, 50, 60, 70 and 80 variables are solved. Additionally, the instances with 4 objectives and 20, 30 and 40 variables are solved.

ii) Uncapacitated facility location problems (UFLP) benchmark instances from Forget et al. (2022b). The instances with 3 objectives and 56, 72 and 90 variables are considered, as well as the instances with 4 objectives and 42 and 56 variables.

iii) Capacitaded facility location problems (CFLP) instances from An et al. (2022) and Bauß and Stiglmayr (2023c). We consider instances with 3 objectives and 65, 119 and 230 variables.

iv) Generalized assignment problems (GAP) test instances from Bauß and Stiglmayr (2023c). The instances with 3 objectives and 48, 75 and 108 variables are solved. Additionally, the instances with 4 objectives and 48 and 75 variables ware solved.

Remarks Regarding the Implementation Due to numerical difficulties, we slightly adapt the implementation, as such it slightly differs from the presented approaches. The first change affects the inequalities, obtained by solving a weighted sum scalarization to integer optimality (cf. Section 5.2.1 and Section 5.2.3). Originally, all constraint coefficients of the considered instances are integer. Nevertheless, the additional obtained inequalities, which are added to the corresponding subproblems, contain in general non-integer coefficients. Bensolve, which we use to compute our lower bound set, relies on the GLPK solver. Unfortunately, for harder and larger problems, the solver faces numerical issues and aborts the run of the algorithm in the worst case. To overcome this issues we round the constraint coefficients in the following way.

Let $\bar{a}_1 x_1 +, \dots, +\bar{a}_n x_n \geq \bar{b}$, with $\bar{a} \in \mathbb{R}^n$ and $\bar{b} \in \mathbb{R}$, be an inequality obtained during the algorithm. Then, the modified inequality is given by $\lceil \bar{a}_1 \rceil x_1 +, \dots, \lceil +\bar{a}_n \rceil x_n \geq \lfloor \bar{b} \rfloor$. Obviously, this constraint is weaker in general, but it is necessary to overcome the numerical issues.

The second change also concerns the usage of the simple lower bounds (cf. Section 5.2.3). The main motivation of using this simple lower bounds is to save computation time by not using Bensolve to compute the complete lower bound set which may consist of a large number of facets. But although we use CPLEX to solve just a single scalarization to integer optimality instead of the complete lower bound set, this is unfortunately much slower in the majority of the considered instances. This is due to the rather slow interface to CPLEX in Julia. Especially the problem building takes a relatively large amount of time. To partially overcome this problem, we limit the CPLEX computation time to $\frac{1}{10}$-th of the time needed to compute the lower bound with Bensolve in the root node. If the program is not solved to optimality within the the given time, the best feasible solution found so far is treated like the optimal solution to possibly update the upper bound and the current best lower bound can be used as simple lower bound. Therefore, it is also possible to use objective space information, although they might be worse. If no feasible solution is found within the time limit and infeasibility is not proven, no additional objective information can be added to the corresponding node.

(KP), $p = 3, n = 40$				
approach	nodes	time (s)	IPs	sol.
BB	138365.8	97.55	0.0	10
NS(LHG)	49898.4	76.85	0.0	10
WST	48157.8	73.00	4.0	10
EC	45384.2	68.99	39.3	10
SLB	51869.0	100.17	9203.5	10
EC+TE	41948.0	68.76	38.4	10
SLB+TE	33993.0	106.21	6838.4	10

(a) Knapsack problem with $n = 40$ variables and $p = 3$ objectives.

(KP), $p = 3, n = 50$				
approach	nodes	time (s)	IPs	sol.
BB	391170.2	398.55	0.0	10
NS(LHG)	112975.4	317.70	0.0	10
WST	110409.0	299.95	4.0	10
EC	101027.0	254.48	50.6	10
SLB	97466.2	391.33	17479.4	10
EC+TE	98385.4	256.40	49.4	10
SLB+TE	94661.0	375.52	17492.8	10

(b) Knapsack problem with $n = 50$ variables and $p = 3$ objectives.

(KP), $p = 3, n = 60$				
approach	nodes	time (s)	IPs	sol.
BB	1201949.8	1794.81	0.0	10
NS(LHG)	321465.3	2591.44	0.0	8
WST	320631.0	2550.18	4.0	9
EC	313461.2	2103.35	62.1	10
SLB	247905.9	1946.32	45446.9	10
EC+TE	279708.3	1496.39	63.1	10
SLB+TE	276558.0	2082.41	49992.1	10

(c) Knapsack problem with $n = 60$ variables and $p = 3$ objectives.

(KP), $p = 3, n = 70$				
approach	nodes	time (s)	IPs	sol.
BB	2470128.8	4312.96	0.0	7
NS(LHG)	447631.9	4533.68	0.0	7
WST	447604.2	4489.07	4.0	7
EC	432262.9	4156.09	67.2	7
SLB	440161.9	4532.67	79311.0	7
EC+TE	439530.9	3828.58	67.8	8
SLB+TE	436677.6	4572.73	79311.2	8

(d) Knapsack problem with $n = 70$ variables and $p = 3$ objectives.

(KP), $p = 3, n = 80$				
approach	nodes	time (s)	IPs	sol.
BB	3433775.1	7200.00	0.0	0
NS(LHG)	376535.2	7200.00	0.0	0
WS	392186.7	7200.00	4.0	0
EC	429098.6	7200.00	88.8	0
SLB	500545.7	7158.29	87007.5	1
EC+TE	418424.2	7200.00	89.1	0
SLB+TE	505265.6	7105.14	88435.9	1

(e) Knapsack problem with $n = 80$ variables and $p = 3$ objectives.

(KP), $p = 4, n = 20$				
approach	nodes	time (s)	IPs	sol.
BB	9073.6	7.76	0.0	10
NS(LHG)	4398.0	6.19	0.0	10
WST	4230.8	6.33	5.0	10
EC	4185.6	7.12	38.7	10
SLB	4289.8	15.57	752.5	10
EC+TE	899.0	4.90	27.7	10
SLB+TE	742.8	8.73	303.4	10

(f) Knapsack problem with $n = 20$ variables and $p = 4$ objectives.

(KP), $p = 4, n = 30$				
approach	nodes	time (s)	IPs	sol.
BB	53531.0	90.03	0.0	10
NS(LHG)	21621.0	78.95	0.0	10
WST	19798.3	68.42	5.0	10
EC	19107.2	67.78	59.2	10
SLB	26117.0	206.31	4699.9	10
EC+TE	15618.0	73.85	59.3	10
SLB+TE	14554.0	135.63	3428.7	10

(g) Knapsack problem with $n = 30$ variables and $p = 4$ objectives.

(KP), $p = 4, n = 40$				
approach	nodes	time (s)	IPs	sol.
BB	325416.1	2070.93	0.0	8
NS(LHG)	115926.3	1815.93	0.0	8
WST	114715.6	1562.56	5.0	8
EC	110539.0	1511.55	72.4	8
SLB	85321.0	1410.12	14958.6	8
EC+TE	111540.9	2235.66	68.6	8
SLB+TE	73076.1	2974.74	11223.3	8

(h) Knapsack problem with $n = 40$ variables and $p = 4$ objectives.

Table 5.1: Numerical results on multi-objective knapsack instances of Kirlik and Sayın (2014).

Results of the Knapsack Problems The numerical results show that the dynamic node selection strategy based on the local hypervolume gap has a large impact on the average number of explored nodes. Note that preliminary tests on a different set of knapsack problems have shown, that this dynamic strategy works better than the strategy based on the search zone box. With the chosen node selection strategy we can reduce the number of explored nodes by up to 71.1% (Table 5.1b), in problem sizes where all 10 instances are solved, and up to 81.9% (Table 5.1d) in instance sizes where the same amount of problems is solved. Since the computation of the local hypervolume gap can be expensive, the total computation time can only be reduced by up to 21.6% (Table 5.1a). Note, that there are also cases, where the total computation time increases, although the number of explored nodes decreases (Table 5.1c and 5.1d). It is not surprising, that this occurs in the instances with a larger amount of variables, since there are possibly more non-dominated points and corresponding local upper bounds, which need to be considered during the gap computation.

The approaches WST and EC reduce the average number of explored nodes in the majority of the considered instances. Nevertheless, the impact on the performance is more significant when EC is used. The usage of SLB can, especially for instances with a larger amount of variables, reduce the computation time and the number of explored nodes. In Table 5.1e, it is shown, that using SLB allows us to solve more instances in the given time limit, which is an improvement compared to the other approaches. Nevertheless, especially for smaller instance sizes, using SLB increases the computational time a lot. This is due to the fact, that the CPLEX interface is rather slow in Julia.

The best working approaches for the considered benchmark instances are EC+TE and SLB+TE. Regarding the average number of explored nodes SLB+TE seems to be the best choice, since this approach creates the least amount of nodes in the majority of the experiments. It is possible to reduce the number of nodes by up to 91.8% (Table 5.1f), respectively 79.4% (Table 5.1c) if we omit the instance size $p = 4, n = 20$, since it highly benefits from the enumeration. The best choice for KP regarding the total computation time seems to be EC+TE, since it is the fastest approach in the majority of the solved instances. The runtime can be reduced by up to 36.9% (Table 5.1f), respectively 36.2% (Table 5.1b).

Results of the Uncapacitated Facility Location Problems Similar to the results of KP, the dynamic node selection strategy has a significant impact on the number of explored nodes and the total computation time. In contrast to the knapsack problems, preliminary tests on a different set of UFLP have shown, that the dynamic node selection strategy based on the search zone box works better. The reason for this might be the larger amount

(UFLP), $p = 3, n = 56$				
approach	nodes	time (s)	IPs	sol.
BB	152349.4	216.07	0.0	10
NS(HSZ)	100222.6	217.16	0.0	10
WST	97243.4	204.34	4.0	10
EC	97649.4	205.35	39.0	10
SLB	95000.8	258.92	17888.5	10
WST+TE	96318.8	216.03	4.0	10
SLB+TE	95795.9	318.66	18159.4	10

(a) Uncapacitated facility location problem with $n = 56$ variables and $p = 3$ objectives.

(UFLP), $p = 3, n = 72$				
approach	nodes	time (s)	IPs	sol.
BB	482483.6	1321.29	0.0	10
NS(HSZ)	277068.6	1249.20	0.0	10
WST	310494.8	1377.52	4.0	10
EC	310802.8	1378.46	42.1	10
SLB	295379.0	1694.93	54184.4	10
WST+TE	310357.8	1310.23	4.0	10
SLB+TE	295245.3	1729.24	54187.9	10

(b) Uncapacitated facility location problem with $n = 72$ variables and $p = 3$ objectives.

(UFLP), $p = 3, n = 90$				
approach	nodes	time (s)	IPs	sol.
BB	1604980.2	7144.28	0.0	2
NS(HSZ)	757425.0	6742.37	0.0	7
WST	712808.9	6328.50	4.0	8
EC	733769.6	6310.58	48.1	8
SLB	715377.1	6814.74	105225.6	6
WST+TE	731303.1	6326.04	4.0	8
SLB+TE	731190.3	6623.71	107625.1	7

(c) Uncapacitated facility location problem with $n = 90$ variables and $p = 3$ objectives.

(UFLP), $p = 4, n = 42$				
approach	nodes	time (s)	IPs	sol.
BB	61048.4	293.36	0.0	10
NS(HSZ)	48404.8	272.20	0.0	10
WST	40328.0	226.39	5.0	10
EC	40947.6	228.46	27.4	10
SLB	42194.3	434.08	7406.1	10
WST+TE	37312.8	273.25	5.0	10
SLB+TE	37446.8	432.76	6918.8	10

(d) Uncapacitated facility location problem with $n = 42$ variables and $p = 4$ objectives.

(UFLP), $p = 4, n = 56$				
approach	nodes	time (s)	IPs	sol.
BB	435160.6	6175.10	0.0	5
NS(HSZ)	313120.4	5745.16	0.0	8
WST	264705.2	4309.92	5.0	10
EC	266740.2	4324.72	34.6	10
SLB	180643.0	5296.01	29414.9	6
WST+TE	262173.8	4721.79	5.0	10
SLB+TE	165112.8	5484.16	30455.25	6

(e) Uncapacitated facility location problem with $n = 56$ variables and $p = 4$ objectives.

Table 5.2: Numerical results for multi-objective uncapacitated facility location instances of Forget et al. (2022b).

of non-dominated points, that cause an even larger amount of local upper bounds. Again, a high amount of local upper bounds increases the time needed to compute the gap between lower and upper bound set. Therefore, the gap measure based on the search zone box performs better, since its computation is way faster. The average number of explored nodes can be reduced by up to 42.2% (Table 5.2b) and the total computation time can be reduced by up to 7.2% (Table 5.2b). Furthermore, disregarding the improvements in explored nodes and runtime, there are also improvements regarding the number of solved instances within the given time limit of two hours. For the instance size $n = 90, p = 3$ the amount of solved instances is improved from 2 to 7 (Table 5.2c) and for the instance size $n = 56, p = 4$ the amount of solved instances is improved from 5 to 8 (Table 5.2e).

The WST method reduces the number of explored nodes and the total computation time, in comparison with NS(HSZ), in nearly all considered instance sizes. Table 5.2b is the only exception. Although EC yields similar results like WST, it is slightly worse in the majority of the benchmark instances. Therefore, there is no need to solve the additional ε-constraint scalarizations, since it is outperformed by WST. Although it seems that the number of explored nodes can be further reduced with SLB, the computation time increases. This even results in a reduced number of solved instances within the time limit (Table 5.2c and Table 5.2e).

Regarding the total computation time WST is the best approach to solve uncapacitated facility problems, since it is the fastest in the majority of the instances. In the best case, the computation time is reduced by up to 30.2% (Table 5.2e), where at the same time the number of solved instances is increased from 5 to 10. Regarding the number of explored nodes WST+TE seems to be a good choice. In the majority of the benchmark instances, this approach yields the lowest number of explored nodes. The average number of explored nodes can be reduced by up to 55.5% in the best case (Table 5.2c).

Results of the Capacitated Facility Location Problems Similar to the uncapacitated facility location problem, preliminary tests on a different set of instances have shown that the dynamic search strategy based on the search zone boxes are computationally more efficient. The number of nodes and the computation time are improved with NS(HSZ). By using the suggested node selection strategy, the number of explored nodes can be reduced by up 30.2% (Table 5.3b) and the total computation time improves by up to 18.9% (Table 5.3b). Using the warmstarting of bound sets (WST) has nearly no impact on the number of explored nodes but the computational time increases. By using EC, the number of nodes is reduced marginally, but there is an improvement regarding the computation time, compared to WST. By also using the simple lower bounds, i.e., using the SLB approach, the number of nodes and the computation time increases. Nevertheless,

(CFLP), $p = 3, n = 65$				
approach	nodes	time (s)	IPs	sol.
BB	59890.6	48.40	0.0	10
NS(HSZ)	49007.6	46.31	0.0	10
WST	48857.0	47.18	4.0	10
EC	48839.6	45.89	20.6	10
SLB	48587.2	55.70	9195.8	10
EC+TE	48839.6	49.50	20.6	10
SLB+TE	47988.4	55.36	9203.3	10

(a) Capacitated facility location problem with $n = 65$ variables and $p = 3$ objectives.

(CFLP), $p = 3, n = 119$				
approach	nodes	time (s)	IPs	sol.
BB	460374.4	1134.55	0.0	10
NS(HSZ)	321311.6	919.95	0.0	10
WST	327431.8	939.44	4.0	10
EC	326869.2	919.65	23.8	10
SLB	345477.6	1009.03	62699.5	10
EC+TE	326869.2	942.93	23.8	10
SLB+TE	345477.6	1012.17	62699.5	10

(b) Capacitated facility location problem with $n = 119$ variables and $p = 3$ objectives.

(CFLP), $p = 3, n = 230$				
approach	nodes	time (s)	IPs	sol.
BB	660929.5	7200.00	0.0	0
NS(HSZ)	358055.8	7200.00	0.0	0
WST	362319.4	7200.00	0.0	0
EC	387534.1	7200.00	16.8	0
SLB	453839.3	7200.00	80912.8	0
EC+TE	391868.6	7200.00	16.8	0
SLB+TE	427363.5	7200.00	83468.0	0

(c) Capacitated facility location problem with $n = 230$ variables and $p = 3$ objectives.

Table 5.3: Numerical results for multi-objective capacitated facility location instances of An et al. (2022) and Bauß and Stiglmayr (2023c).

compared to our baseline implementation (BB) there are improvements regarding the number of nodes and there can also be an improvement regarding the runtime.

The best approach regarding the computational time seems to be EC. In the best case, the time can be reduced by 18.9% (Table 5.3b). Regarding the number of explored nodes, there seems to be no consistency caused by the small amount of benchmark instances. NS(HSZ), SLB+TE and EC seem to be good choices to decrease the number of explored nodes, where those can be reduced by 30.2% in the best case.

Results of the Generalized Assignment Problems Preliminary numerical tests on a different test set show, that the node selection with hypervolume of the search zone box NS(HSZ) works better than the one with the local hypervolume gap NS(LHG). This node selection strategy has a significant impact on the number of explored nodes and on the computational time as well. The number of nodes is reduced by up to 53.4% (Table 5.4c), while the average runtime is reduced by up to 38.9% (Table 5.4c).

We can observe that in the tri-objective instances WST outperforms EC, but in the benchmark instances with four objectives EC outperforms WST. Therefore, the best choice regarding the computation time for instances with three objectives is WST, which reduces the computation time by up to 40.5% (Table 5.4c). For the instances with four objectives EC seems to be the best choice regarding the computation time with a reduction of up to

(GAP), $p = 3, n = 48$				
approach	nodes	time (s)	IPs	sol.
BB	27890.0	16.76	0.0	10
NS(HSZ)	20608.0	14.57	0.0	10
WST	18458.4	13.17	4.0	10
EC	18506.4	13.96	32.7	10
SLB	18567.8	20.51	3516.7	10
WST+TE	17469.0	14.08	4.0	10
EC+TE	17501.0	14.91	34.3	10
SLB+TE	17895.8	21.27	3336.5	10

(a) Generalized assignment problem with $n = 48$ variables and $p = 3$ objectives.

(GAP), $p = 3, n = 75$				
approach	nodes	time (s)	IPs	sol.
BB	170359.4	220.20	0.0	10
NS(HSZ)	98703.4	162.00	0.0	10
WST	95179.0	154.56	4.0	10
EC	95760.4	157.35	44.5	10
SLB	66464.0	119.34	12716.6	10
WST+TE	94572.0	166.30	4.0	10
EC+TE	95134.0	168.09	45.1	10
SLB+TE	61568.0	113.74	11676.6	10

(b) Generalized assignment problem with $n = 75$ variables and $p = 3$ objectives.

(GAP), $p = 3, n = 108$				
approach	nodes	time (s)	IPs	sol.
BB	1041528.2	2566.35	0.0	10
NS(HSZ)	484977.8	1569.04	0.0	10
WST	467268.8	1527.91	4.0	10
EC	469200.1	1534.35	53.7	10
SLB	387682.2	1603.55	74288.6	10
WST+TE	464769.0	1632.80	4.0	10
EC+TE	466327.2	1650.48	55.3	10
SLB+TE	367327.0	1746.80	68916.3	10

(c) Generalized assignment problem with $n = 108$ variables and $p = 3$ objectives.

(GAP), $p = 4, n = 48$				
approach	nodes	time (s)	IPs	sol.
BB	145278.4	529.94	0.0	10
NS(HSZ)	99512.2	494.75	0.0	10
WST	85777.2	436.29	5.0	10
EC	77102.8	325.46	26.6	10
SLB	74366.3	386.94	13771.3	10
WST+TE	78313.0	494.43	5.0	10
EC+TE	78622.2	437.83	29.0	10
SLB+TE	73206.1	797.21	13150.0	10

(d) Generalized assignment problem with $n = 48$ variables and $p = 4$ objectives.

(GAP), $p = 4, n = 75$				
approach	nodes	time (s)	IPs	sol.
BB	836528.8	6292.90	0.0	4
NS(HSZ)	490997.9	6185.76	0.0	5
WST	466591.2	6128.25	5.0	5
EC	466653.3	5984.18	33.4	5
SLB	297399.0	6423.18	54521.0	5
WST+TE	463480.6	6013.66	5.0	6
EC+TE	466678.0	6004.96	35.0	6
SLB+TE	282099.1	6925.44	55286.5	5

(e) Generalized assignment problem with $n = 75$ variables and $p = 4$ objectives.

Table 5.4: Numerical results for multi-objective generalized assignment instances of Bauß and Stiglmayr (2023c).

38.6% (Table 5.4d). Regarding the average number of explored nodes SLB+TE performs best, since it explores the fewest nodes in the majority of the considered instances. In the best case, the number of nodes is reduced by 64.7% (Table 5.4e), considering the instance sizes where all problems are solved. If we look at the results in Table 5.4e, we even see a reduction of nodes by 66.3% and a simultaneous improvement in the number of solved instances. Nevertheless, there are other approaches for this instance size, that solve even more problems.

Summary and General Observations In all of the four tested problem classes (KP, UFLP, CFLP and GAP), a significant reduction of the average number of explored nodes and the average total computation time can be realized in nearly every tested adapted branch and bound approach. With a rising number of variables the impact on the performance increases, whereas a rising number of objective functions results in a decrease of the performance. This is due to the general shortcomings of multi-objective branch and bound, struggling with weaker bounds in higher dimensions. In Figure 5.2, performance profiles of the corresponding best approaches are illustrated. Thereby, the x-axis represents the time in seconds and the y-axis corresponds to the proportion of solved instances.

In three of the four considered problem classes, preliminary tests have shown, that the dynamic node selection strategy based on the search zone box works better in terms of computation time compared to the local hypervolume gap strategy. Only for knapsack problems the local hypervolume gap is used. The stronger combinatorial structure of UFLP, CFLP and GAP (compared to KP) leads to more complex lower bound sets, which are computationally difficult to handle in the local hypervolume gap strategy.

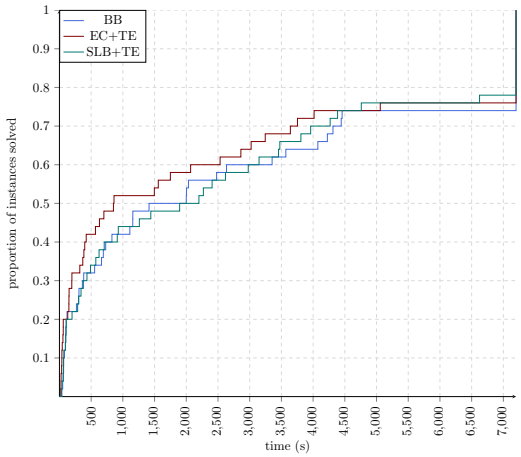
Unfortunately, there is no clear winner that performs best on all considered benchmark instances, but we can observe some tendencies. Regarding the number of nodes, SLB+TE seems to be a good choice. For every considered problem class, SLB+TE performs comparatively good and is even the best choice in some of them. This approach can also improve the number of solved instances in a few cases. But, due to the high amount of solved integer programming scalarizations this approach often increases the total computation time, especially for smaller instance sizes. Regarding the total computation time, WST and EC seem to be good choices. Both approaches are the best performing methods in some of the problem classes and perform also comparatively good in the other ones.

Nevertheless, by just using the corresponding dynamic node selection strategy, remarkable improvements are achieved with respect to both — the number of considered branch and bound nodes and the computation time.

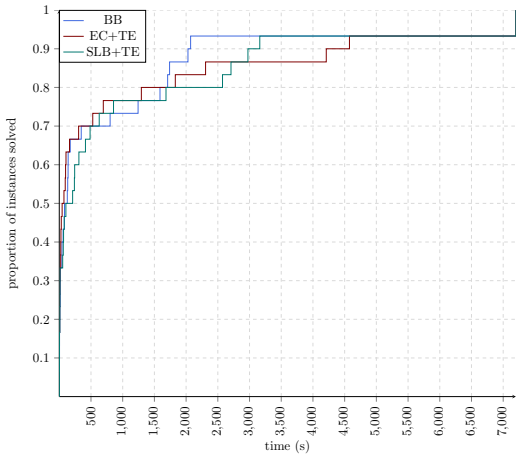
To conclude this chapter, we summarize its major contributions and findings. We propose different adaptive improvements for multi-objective branch and bound frameworks

using objective-space information to partially overcome its structural difficulties. We propose new dynamic node selection strategies, based on the multidimensional gap between lower and upper bound set, which improve the number of explored nodes and the total computation time. Furthermore, we use objective space information gained by solving scalarized (sub)problems to integer optimality to improve the lower and upper bound set. A warmstarting approach for the bound sets is proposed and partial enumeration is applied. Additionally, simple lower bound sets are used adaptively to omit the computation of the complete lower bound set. The numerical results show the positive impact on different problem classes regarding the number of explored nodes and the computational time.

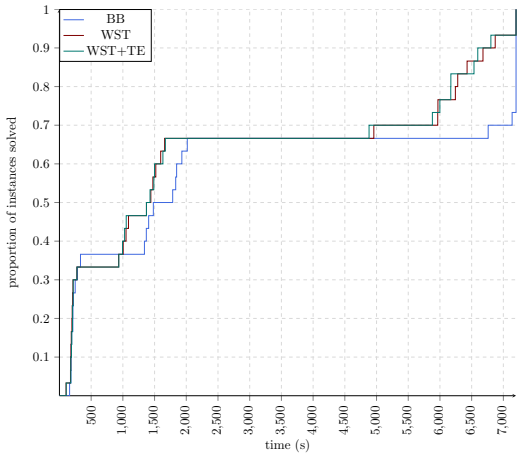
This chapter shows, that the order in which the nodes are explored, has a significant impact on the performance of multi-objective branch and bound algorithms. Since the order depends on the node selection strategy and the branching rule, it might be promising to include different combinations of those components. Especially the combination of different dynamic, problem dependent strategies could result in further improvements.



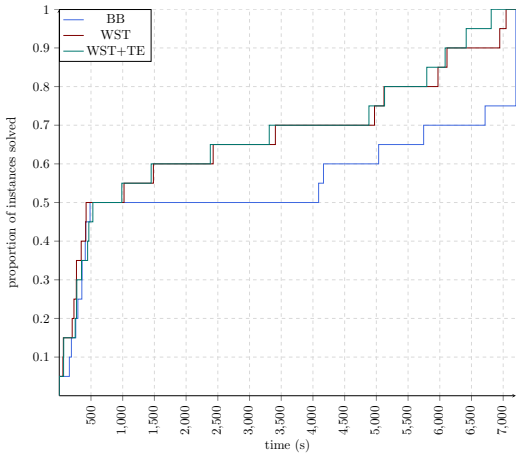
(a) Performance of tri-objective knapsack problems.



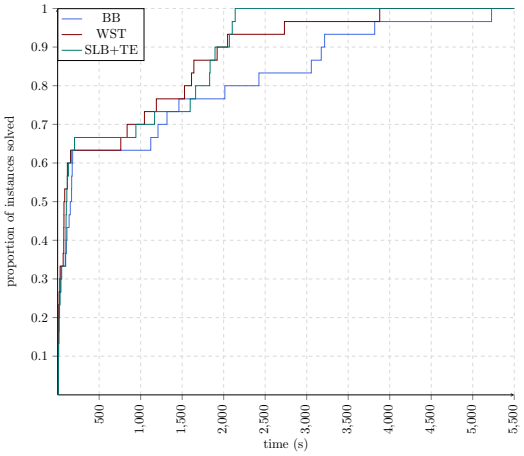
(b) Performance of 4-objective knapsack problems.



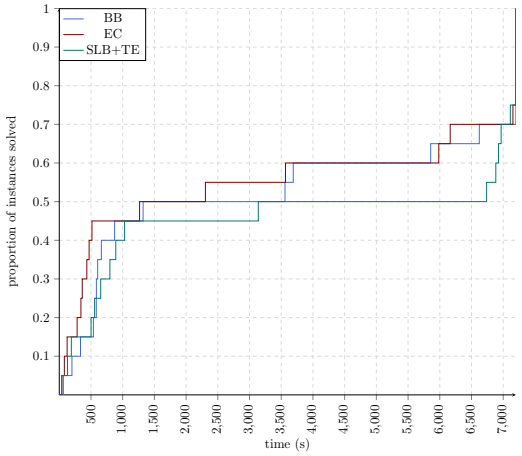
(c) Performance of tri-objective uncapacitated facility location problems.



(d) Performance of 4-objective uncapacitated facility location problems.



(e) Performance of tri-objective generalized assignment problems.



(f) Performance of 4-objective generalized assignment problems.

Figure 5.2: Performance profiles of selected approaches regarding different problem classes.

6 Branching and Queuing for Multi-objective Branch and Bound

In the previous chapters, we elaborated the importance of the order in which the nodes are explored. A sequencing of subproblems can have significant impact on the performance of a multi-objective branch and bound. Thereby, it is crucial not only to find efficient solutions as soon as possible but also to find a set of (efficient) solutions whose images are well distributed along the non-dominated frontier — in other words to find a good representation of the non-dominated set. In this chapter we evaluate the performance of multi-objective branch and bound algorithms depending on the branching and queuing of subproblems. Note that both components affect each order. We use, e.g., the hypervolume indicator as a measure for the gap between lower and upper bound set to implement a multi-objective best-first strategy. We test and evaluate our approaches on different problem classes.

Contribution The majority of this paper is published in Bauß and Stiglmayr (2024a).

Organization of the Chapter The remaining chapter is organized as follows. In Section 6.1, we discuss some of the most common branching rules and node selection strategies of multi-objective branch and bound frameworks. We also give an illustrative example of different ways to measure the gap between lower and upper bound. In Section 6.2, we test all combinations of the presented approaches on three different multi-objective problem classes, namely knapsack, uncapacitated facility location and generalized assignment problems with $p = 3$ objectives.

6.1 Sequencing of Subproblems

In the introduction to multi-objective branch and bound algorithms (cf., Section 3.2.2), we already gave a brief introduction into different branching rules and node selection strategies. Based on this, we give more detailed and formal definitions of the considered node selection and queuing strategies. Thereby, we restrict ourselves to the decision space branching.

Firstly, we introduce some of the common branching rules. We focus on a selection of well known approaches and distinguish them between static and dynamic rules. Secondly, we formally introduce static and dynamic node selection strategies reconstructing the best-first strategy in multiple objectives. Thereby, we mainly focus on different ways to measure the gap between the lower and upper bound.

6.1.1 Multi-objective Branching Rules

In the binary case, decision space branching rules are restricted to the selection of an appropriate branching variable. Therefore, it is necessary to apply a predefined branching rule. As already mentioned in previous sections, we can distinguish between static and dynamic strategies.

Static strategies fix a predefined order of the variables. In every iteration, the first variable of this order, that is not fixed yet in the corresponding subproblem, is chosen. Thus, static strategies are not able to adapt to the problem structure during the run of the algorithm using information obtained in previous iterations. The most basic approach is choosing the branching variable based on the variable indices (either in ascending or descending order). This strategy does not even take problem specific properties into account and is just based on the principle of chance. Thus, we do not consider those approaches for the numerical test in Section 6.2.

Many approaches in the literature propose to use a specific order of the ratios between costs and weights for single-objective knapsack problems (Kellerer et al., 2004). For clarification, we recall that a multi-objective knapsack problem can be written in the following form

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} c_i^j \, x_i && j = 1, \dots, p \\ \text{s.t.} \quad & \sum_{i=1}^{n} w_i \, x_i \leq b \\ & x \in \{0,1\}^n. \end{aligned}$$

In the single-objective case ($p = 1$), the profit to weight ratio of a variable x_i is given by c_i^1/w_i. The variables are considered by a decreasing ratio, which implies that first variables are the more promising ones. Since an extension to multiple objectives is not straightforward, there are several different approaches (see, e.g., Bazgan et al., 2009; Jorge, 2010; Ulungu and Teghem, 1997).

Due to the multiple objectives, we define a ratio vector $r^i := (r_1^i, \dots, r_p^i) \in \mathbb{R}^p$ for each variable $x_i, i = 1, \dots, n$ by $r^i := (c_i^j/w_i)_{j=1,\dots,p}$. Bazgan et al. (2009) propose the

branching rule *sum of ratios*, where the entries of the ratio vectors are summed up, i.e., $sr^i := \sum_{j=1}^{p} c_i^j / w_i$ for all $i = 1, \ldots, n$. A free variable x_i with the largest value sr^i is then selected as branching variable. Note that we are considering multi-objective optimization with minimization objectives in this work. Thus, we analogously consider the sum of ratios in an increasing order. In Jorge (2010), a slightly different approach is proposed. Again, the ratio vectors r^i, $i = 1, \ldots, n$ are considered but instead of using the L_1-norm, we check for dominance among them and count how often each vector is dominated. So, the variable with the ratio vector which is least often dominated by other ratio vectors is selected as branching variable. We therefore refer to this as the *dominance of ratios* rule.

Both presented rules based on the ratio vectors were proposed for knapsack problems but the branching scheme can be analogously applied on, e.g., facility location problems and generalized assignment problems, when opening costs and workload of a task are interpreted as the "weight" w_i.

Dynamic branching strategies do not have a predefined order in which the variables are considered. They take additional information into account, which is gained during the algorithm. The choice of the next variable to branch on is therefore dependent on the current node. The following two branching rules rely on the lower bound set $\mathcal{L}$ of a node ν. The so-called *most often fractional* rule counts for every variable x_i in how many extreme points of the lower bound set $\mathcal{L}$ it attains an fractional value. The variable which is most often fractional is chosen as branching variable. Belotti et al. (2013) propose a slightly different approach. Instead of simply counting how often a variable is fractional, they sum for each variable the distances to the next integer solution in all extreme points of the lower bound set $\mathcal{L}$. Then, the variable with the highest total distance is used as branching variable. In the following we call this the *how fractional* rule.

6.1.2 Multi-objective Node Selection Strategies

In each iteration of a branch and bound algorithm, a node has to be selected. Similar to the branching rules, the node selection strategies can be distinguished between static and dynamic strategies.

The most frequently applied static strategies are *depth-first search* and *breadth-first search*. The depth-first strategy selects (if possible) a newly created node, i.e., a child node of the the current active node. If a node can be fathomed and no new nodes are created in that iteration, the lastly created node is selected. Therefore, it follows a *last in first out* approach. Contrary, the breadth-first strategy selects always the node which was created first and has not been explored so far. Thus, this approach is based on a *first in first out* principle. Both variants do not require additional computations, are easy to

implement, but do not adapt to the problem structure.

Dynamic node selection strategies choose the active node depending on the expected improvement of the upper and/or lower bound set. There are several approaches which mimic the best first node selection strategy which is most frequently applied in single-objective branch and bound algorithms (see, e.g., Morrison et al., 2016). Although this is the standard approach in the single-objective case, there is no direct extension to the multi-objective case. However, there are several different gap measures proposed in the literature. Some of them are presented and evaluated in the following. In these strategies, the node with the largest gap is chosen as it is assumed that a large gap coincides with a large potential of improvement. The methods mainly differ in the way the gap between the lower and upper bound set is measured. Note that although the idea is to measure the gap between lower and upper bound set, often the gap is measured between the lower bound set $\mathcal{L}$ and the set of local upper bounds $\mathcal{D}(\mathcal{U})$. This distance represents the "open space" in which additional non-dominated points might be located.

The first method is the *local hypervolume gap* strategy which is proposed for the bi-objective case in Section 4.1 and is extended to the multi-objective case in Section 5.1. In Jesus et al. (2021), the impact of the exact hypervolume gap on the performance of multi-objectve branch and bound is evaluated. However, the numerical results show that the evaluation of the hypervolume is computationally so demanding that its positive effects on the number of created nodes are compensated in terms of runtime. To avoid this computational effort the hypervolume of the largest search zone is approximated by the volume of the simplex spanned by the local upper bound (cf. Section 5.1) and the intersections of the extreme rays of its dominance cone with the lower bound set $\mathcal{L}$. Similarly, the largest *hypervolume of a search-zone* box spanned by a local upper bound and the local ideal point of the lower bound set (cf. Section 5.1) can be used as a gap measure.

The well known *Hausdorff distance* is also applied as measure of the gap between lower bound and the upper bound. This approach has been recently used by Adelgren and Gupte (2022) for bi-objective branch and bound approaches. The (directed) Hausdorff distance for a node ν from the set of local upper bounds $\mathcal{D}(\mathcal{U})$ to the lower bound set $\mathcal{L}$ is given by

$$hd(\nu) \coloneqq \max_{u \in \mathcal{D}(\mathcal{U})} \min_{\substack{l \in \mathcal{L} \\ l \leqq u}} d(u, l),$$

where $d(u, l)$ is the L_2-distance between two points $u, l \in \mathbb{R}^p$.

The so-called *width of enclosure*, proposed in Eichfelder et al. (2021), is a similar method to measure the gap. It only differs in the chosen distance measure between the two points.

Instead of the L_2-distance the minimal componentwise distance is used. Therefore, the width of enclosure of a node ν is given by

$$woe(\nu) \coloneqq \max_{u \in \mathcal{D}(\mathcal{U})} \min_{\substack{l \in \mathcal{L} \\ l \leqq u}} \min_{j=1,\dots,p} (u_j - l_j)$$

All the above presented gap measurements are illustrated in Figure 6.1 for a bi-objective example.

Note that when a node is created in the algorithm we assign the gap of its parent node to it to avoid the computational overhead of precomputing lower bounds. Moreover, the gap would have to be updated in every node when the upper bound set changes. This is computationally very demanding and the potential reduction in number of nodes cannot compensate for this.

6.2 Numerical Tests

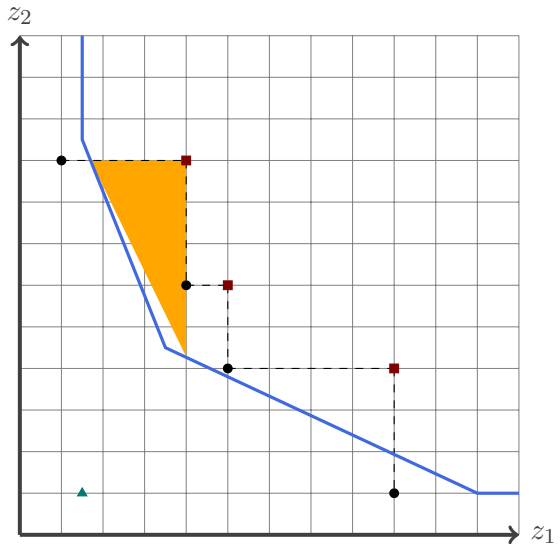
All the algorithms were implemented in Julia 1.9.0 and the linear relaxations (for the lower bound set) were solved with Bensolve 2.1 (Löhne and Weißing, 2017). The numerical test runs were executed on a single core of a 3.20 GHz Intel® Core™ i7-8700 CPU with 32 GB RAM. The number of nodes and computation times are average values over 10 instances.

We present numerical results of all combinations of the branching rules and node selection strategies, that were introduced in the previous section. Thereby, we consider three different tri-objective problem classes:

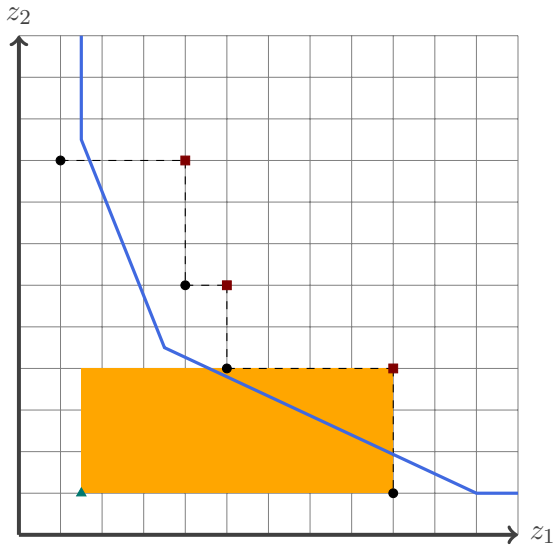
- Knapsack problems (KP) with 3 objectives and 30, 40 and 50 variables. The instances are extracted from Kirlik and Sayın (2014).
- Uncapacitated facility location problems (UFLP) with 3 objectives and 42, 56 and 72 variables. We use the instances of Forget et al. (2022b).
- Generalized assignment problems (GAP) with 3 objectives and 27, 48 and 75 variables. The instances were randomly generated and are publically available in Bauß and Stiglmayr (2023c).

To ease notation we define abbreviations for the tested branching strategies and node selection rules in Table 6.1.

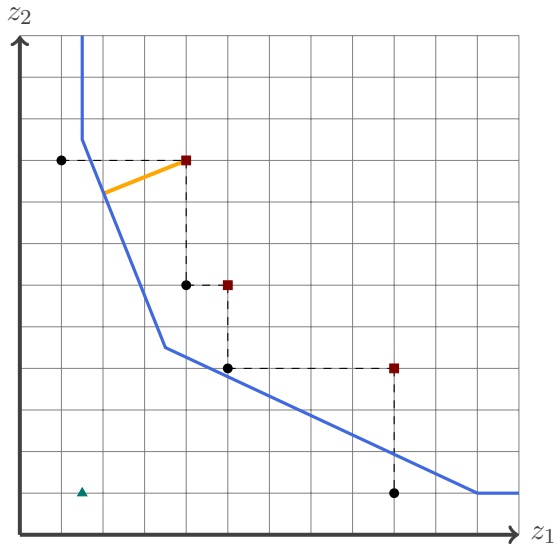
In Table 6.2, the numerical results for the tri-objective knapsack problem are shown. The numbers in brackets indicate how many (if not all) of the instances have been solved in the time limit of one hour. For all considered instance sizes the combination of *local hypervolume gap* node selection and *how fractional* branching (HVG-HF) is the best choice



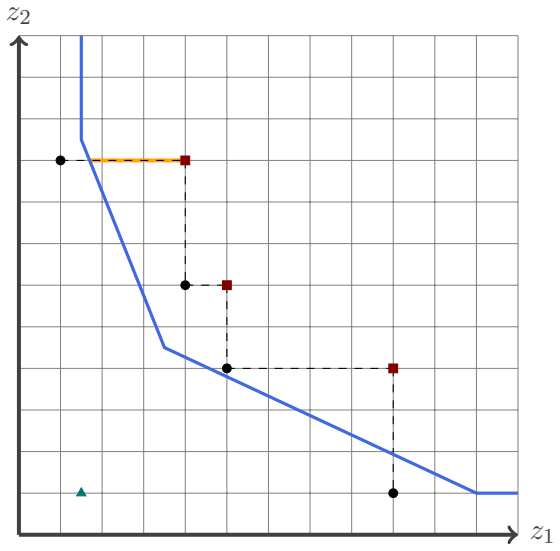
(a) The yellow triangle illustrates the local hypervolume gap of this example.



(b) The yellow box indicates the hypervolume of a search-zone box.



(c) The Hausdorff distance between lower bound and upper bound is depicted by the yellow line.



(d) The yellow line indicates the width of enclosure in the corresponding node.

Figure 6.1: An exemplary illustration of different gap measurements for a bi-objective (MO01LP).

node selection	
DF	depth-first
BF	breadth-first
HVG	local hypervolume gap
HVB	search-zone box
HD	Hausdorff distance
WOE	width of enclosure

branching rule	
MOF	most often fractional
HF	how fractional
SR	sum of ratios
DOM	dominance of ratios

Table 6.1: Abbreviations for node selection strategies and branching rules.

(KP)	$p = 3, n = 30$		$p = 3, n = 40$		$p = 3, n = 50$	
	nodes	time(s)	nodes	time (s)	nodes	time (s)
DF-MOF	23985.0	12.8004	138368.4	111.1310	391170.2	438.8650
DF-HF	25386.8	13.5571	145034.2	115.6625	390635.6	436.7868
DF-SR	13251.6	7.5527	51868.0	41.9066	116711.2	135.2845
DF-DOM	13562.6	7.7311	55519.6	44.8569	123384.0	144.2920
BF-MOF	22689.6	17.7095	149388.0	196.9730	432790.0	757.8097
BF-HF	21375.0	16.2624	134515.2	179.8361	407798.4	739.3253
BF-SR	341674.8	200.2637	1269261.8	1256.6632	1433460.3 (5)	3187.8977 (5)
BF-DOM	329000.8	192.8351	1234688.4	1222.4330	1525962.0 (4)	3393.6130 (4)
HVG-MOF	10355.4	9.4816	49898.4	88.4784	112975.4	334.5662
HVG-HF	9886.0	8.7751	49432.8	81.3003	109233.2	307.8950
HVG-SR	27234.4	28.2833	101070.4	215.4569	206470.0	913.4311
HVG-DOM	24432.4	25.3734	93457.2	199.2275	196919.6	873.1798
HVB-MOF	14355.6	11.1059	94993.8	146.8714	233845.0	483.6893
HVB-HF	14241.4	10.4059	91549.2	130.7214	230877.6	462.5850
HVB-SR	36433.6	32.3518	151624.6	229.3973	325188.8 (9)	930.8374 (9)
HVB-DOM	35701.8	31.7019	148932.2	225.3204	348987.4 (9)	998.9598 (9)
HD-MOF	12462.2	9.6071	79112.4	126.2463	221310.9 (9)	821.2436 (9)
HD-HF	11919.6	9.0139	78892.8	127.9482	241806.8	826.3442
HD-SR	38704.4	38.9071	191524.8	571.2338	352888.6 (7)	2144.3936 (7)
HD-DOM	33764.6	33.9414	162471.4	484.5804	335842.2 (7)	2040.8080 (7)
WOE-MOF	15256.2	10.5954	82633.0	99.8712	238207.2	423.3763
WOE-HF	15068.8	10.2355	83082.8	101.3736	239090.6	434.7994
WOE-SR	37537.6	25.3568	264445.2	337.7509	721534.2	1628.2839
WOE-DOM	28092.8	18.9768	184564.6	235.7279	563025.8 (9)	1270.5786 (9)

Table 6.2: Tri-objective knapsack problems with 30, 40 and 50 variables (Kirlik and Sayın, 2014).

(UFLP)	$p = 3, n = 42$		$p = 3, n = 56$		$p = 3, n = 72$	
	nodes	time(s)	nodes	time (s)	nodes	time (s)
DF-MOF	32880.8	38.8117	139834.2	224.0030	418528.6	1467.8659
DF-HF	36586.6	41.1587	145078.8	228.3968	467150.8	1584.7095
DF-SR	41969.0	41.6262	191415.4	268.0743	615405.2	1867.3283
DF-DOM	35405.8	39.2697	196196.0	282.8492	587028.4	1746.0606
BF-MOF	24487.8	33.5881	95060.2	248.2963	258674.4	1233.3230
BF-HF	26940.8	35.8490	100479.6	257.3975	281206.2	1335.8392
BF-SR	31165.4	35.7170	114821.8	269.1075	297056.4	1292.9423
BF-DOM	22001.2	29.1760	98421.6	238.5091	254647.0	1127.5445
HVG-MOF	23870.6	51.1781	91803.4	443.0579	233284.8	2780.9655
HVG-HF	26221.4	54.3288	96871.2	462.4322	252660.5	2957.6853
HVG-SR	30934.2	53.1506	112207.2	454.0694	279531.1	2726.9569
HVG-DOM	21482.8	42.3663	95210.4	404.7764	244279.6	2361.3996
HVB-MOF	24519.4	33.8085	95308.0	219.6553	255510.4	1298.5159
HVB-HF	26928.2	35.7254	99950.6	230.4688	278603.4	1359.0283
HVB-SR	31324.0	37.3422	114875.0	262.8621	298007.4	1384.1356
HVB-DOM	22179.4	29.7234	99157.2	211.9494	254735.6	1154.9444
HD-MOF	31687.4	40.3926	136396.2	256.0803	405774.0	1636.6743
HD-HF	35171.6	42.7147	141459.8	260.2738	455097.0	1752.7722
HD-SR	41224.0	43.8607	185167.4	309.8969	592779.0	2098.8174
HD-DOM	33231.8	40.0666	187920.8	334.2814	551169.4	1900.0103
WOE-MOF	27435.6	51.6598	114170.0	415.1545	319554.0	2751.3246
WOE-HF	30474.4	54.2606	117135.4	419.9718	344775.2	2917.7876
WOE-SR	34836.4	52.9098	139033.4	419.5647	377158.3	2720.1222
WOE-DOM	25593.0	44.2695	122627.0	394.0369	343180.7	2450.4333

Table 6.3: Tri-objective uncapacitated facility location problems with 42, 56 and 72 variables (Forget et al., 2022b).

with respect to the number of created branch and bound nodes. However, due to the relatively costly computation of the local hypervolume gap, it does not lead to the best computation times. Although the combination of *depth-first search* with the *sum of ratios* branching rule (DF-SR) creates more nodes than HVG-HF, it is the best choice in terms of the total computation time for all considered instance sizes. Note, that the static combinations of BF-SR and BF-DOM perform considerably worse regarding number of nodes and computation time in each problem size. If we consider the respectively best approaches and compare them to DF-MOF, which is chosen in our baseline branch and bound framework, we can reduce the number of nodes by up to 72.1% and the total computation time by up to 69.2%.

In Table 6.3, the numerical results for the tri-objective uncapacitated facility location problems are shown. Regarding the number of explored nodes, no combination of node selection strategy and branching rule shows to be clearly superior to the others. But the *local hypervolume gap* in combination with the *most often fractional* (HVG-MOF) and *dominacne of ratios* rule (HVG-DOM) are the most promising. In the best case, we achieve a reduction of explored nodes by up to 44.3% compared to the basic approach DF-MOF. Regarding the computational time the combinations BF-DOM and HVB-DOM yield the best results since they perform good all considered instance sizes. We can reach

a reduction of computation time by up to 24.8%. Note that both considered combinations rely on the static dominance of ratios branching rule.

(GAP)	$p = 3, n = 27$		$p = 3, n = 48$		$p = 3, n = 75$	
	nodes	time(s)	nodes	time (s)	nodes	time (s)
DF-MOF	2145.2	0.8989	25214.0	22.7623	150039.4	243.4742
DF-HF	2312.8	0.9342	25624.6	23.0392	148997.8	237.6341
DF-SR	2728.8	1.0449	31214.0	25.7561	187934.2	306.8806
DF-DOM	2814.2	1.0776	28019.0	23.1198	150022.8	227.3948
BF-MOF	1794.2	0.7826	17324.0	18.0416	83273.4	162.8612
BF-HF	1926.0	0.8348	17614.2	18.6211	85289.2	174.1150
BF-SR	2472.4	1.0170	23418.4	22.9168	113958.0	230.9721
BF-DOM	2617.4	1.0767	20730.6	20.2865	97610.8	188.5939
HVG-MOF	1778.4	0.8929	17030.6	22.7282	80720.2	276.1176
HVG-HF	1910.8	0.9563	17330.4	23.0759	82971.2	283.6493
HVG-SR	2430.2	1.1571	23147.0	28.0784	111541.8	405.7946
HVG-DOM	2585.2	1.2309	20442.2	24.7974	95689.8	320.4952
HVB-MOF	1802.0	0.8076	17285.4	17.9048	83615.4	166.1450
HVB-HF	1927.8	0.8504	17608.8	18.1974	85378.8	170.1354
HVB-SR	2450.8	1.0291	23538.2	22.3729	113930.0	228.2173
HVB-DOM	2609.6	1.0958	20879.0	19.8453	98906.0	188.6762
HD-MOF	2014.2	0.8762	23257.2	22.6969	132736.8	241.9842
HD-HF	2185.6	0.9402	23755.2	23.0600	129838.2	229.5523
HD-SR	2583.0	1.0513	30504.8	27.5358	172742.2	306.2228
HD-DOM	2716.8	1.1058	26249.4	23.6946	132324.4	223.6301
WOE-MOF	1891.6	0.9198	18414.0	23.1183	91496.4	280.4083
WOE-HF	2013.8	0.9715	18764.4	23.3688	91927.6	283.7661
WOE-SR	2502.0	1.1550	24475.2	27.8896	123246.4	400.8106
WOE-DOM	2627.6	1.2130	21986.2	25.0534	106589.6	314.8106

Table 6.4: Tri-objective generalized assignment problems with 27, 48 and 75 variables (Bauß and Stiglmayr, 2023c).

Table 6.4 shows the numerical results of the corresponding tri-objective generalized assignment problems. For all tested problem sizes the combination of the *local hypervolume gap* node selection with the *most often fractional* branching rule (HVG-MOF) is the best choice regarding the number of created notes. However, this approach does not achieve the best computation times, since the reduction of nodes does not compensate the relatively high computational costs of the dynamic node selection strategy. Regarding the total computation time, the combination BF-MOF seems to be overall the best choice. Although the time of BF-MOF is outperformed by HVB-MOF for $n = 48$, we still consider BF-MOF as the best choice overall, since the difference is so small (0.76%). Compared to the baseline combination DF-MOF, we can reach a reduction of nodes by up to 46.2% and a decrease of the total computation time by up to 33.1%. Note that all favorable combinations for the general assignment problems use the most often fractional branching rule (MOF).

Concluding, we tested 24 different combinations of decision space branching rules and node selection strategies and compared their results. The numerical tests show that the best approach in each problem class regarding the number of nodes uses the node selection strategy based on the local hypervolume gap. This again shows the huge impact of

this strategy, which is introduced in Section 4.1 and extended to multiple objectives in Section 5.1. Due to its costly computation, this node selection strategy is not the best regarding the computation time, but it still yields competitive run times. Furthermore, the corresponding branching rules are the two dynamic ones — *most often fractional* (MOF) and *how fractional* (HF).

7 Conclusion

In this thesis, we investigate multi-objective branch and bound algorithms. We focus on a selection of its key components and propose numerous approaches and techniques to improve the performance. All approaches and components are described in detail for (MO01LP) but can be easily extended to (MOILP).

We start by augmenting bi-objective branch and bound algorithms. Since the two-dimensional objective space holds certain properties, like the natural order, it is plausible to distinguish between the bi-objective and the multi-objective case with $p \geq 3$ objective functions. We introduce two new node selection strategies, based on different gap measurements between lower and upper bound. Numerical tests show that the local hypervolume gap strategy outperforms the global hypervolume gap strategy. Using the local hypervolume gap node selection strategy, we can reduce the number of created nodes in the branch and bound tree by up to 76% and the computation time by up 73%. This has a huge impact on the performance of bi-objective branch and bound by just changing the node selection strategy. Additionally, we hybridized the branch and bound by adaptively solving scalarizations of the root node problem to integer optimality. This proposed procedure yields objective space information to improve upper and lower bounds which in turn results in a higher fathoming rate. The best performing proposed approach relies on the new local hypervolume gap node selection strategy and the usage of weighted sum and augmented weighted Tchebycheff scalarizations to improve the bound sets. We can reduce the number of nodes by up to 83% and the total computational time by up to 80% compared to the proposed basic multi-objective branch and bound approach.

Since the proposed methods have a considerable impact on bi-objective branch and bound, we extend the most promising approaches to the multi-objective case with $p \geq 3$. Some of the proposed methods have to be reconsidered and redesigned due to the different structure of the non-dominated set. Other methods can be directly extended and adapted. However, their impact on the performance shows to be different. We present two node selection strategies, where one is based on the local hypervolume gap and the other one is based on the hypervolume of a search zone. However, numerical tests show that there is no best strategy among these since their performance is depending on the considered problem class. This is due to the structure of its non-dominated set, namely the number of non-dominated points, the number of local upper bounds and the number of facets of the lower bound sets. In the considered benchmark instances, the number of explored

nodes is reduced by up to 82% and the total computation time is reduced by up to 22% if the node selection strategy based on the local hypervolume gap is used. By choosing the hypervolume of a search-zone box strategy, the number of nodes can be reduced by up to 53% and the computation time is reduced by up to 39%. Furthermore, we present ways to improve the bounds by solving scalarizations to integer optimality. We propose a warmstarting scheme, a way to omit the computation of the complete lower bound set and an adaptive scheme to possibly generate unsupported non-dominated points. With different combinations of these approaches we can reduce the number of explored nodes by up to 82% and the total computation time can be reduced by up to 41%.

After considering the components of node selection and bounds of multi-objective branch and bound, we compare the performance of different static and dynamic branching rules and node selection strategies. This is the first comprehensive study of branching rules and node selection strategies for multi-objective branch and bound in literature. We test 24 different combinations and show that for each considered problem class the best approach regarding the number of nodes uses our proposed node selection strategy based on the local hypervolume gap. This again shows the significant impact of the new node selection strategy.

Nomenclature

$\mathcal{D}(\mathcal{U})$	Set of local upper bounds
$\mathcal{I}(\nu)$	Set of free variables in node ν
cl()	Closure operator
conv()	Convex hull operator
$\mathcal{L}$	Lower bound set
$\mathcal{H}$	Hyperplane seperating a space into two half-spaces
$\mathcal{P}_{\mathcal{H}}$	Half-space representation of polyhedron $\mathcal{P}$
$\mathcal{P}_{VR}$	Vertex-ray representation of polyhedron $\mathcal{P}$
ν	Node in a branch and bound tree
$\mathbb{R}^p_>$	$\{y \in \mathbb{R}^p \colon y > 0\}$
$\mathbb{R}^p_\geq$	$\{y \in \mathbb{R}^p \colon y \geq 0\}$
$\mathbb{R}^p_\geqq$	$\{y \in \mathbb{R}^p \colon y \geqq 0\}$
$\mathcal{U}$	Upper bound set
$A^{\complement}$	Complement of the set A
m	Number of constraints
n	Number of variables
p	Number of objectives
$x \in \mathbb{R}^n$	Solution of an optimization problem
X	Set of feasible solutions
$X_{\mathcal{U}}$	Incumbent list
X_E	Set of efficient solutions
X_{WE}	Set of weakly efficient solutions
Y	Set of points in the objective space with a corresponding feasible solution
Y_N	Set of non-dominated points

Y_{SN1}	Set of supported non-dominated extreme points
Y_{SN2}	Set of supported non-dominated non-extreme points
Y_{SN}	Set of supported non-dominated points
Y_{WN}	Set of weakly non-dominated points
$z(x) \in \mathbb{R}^p$	Objective vector of a solution x

Bibliography

Achterberg, T., T. Koch, and A. Martin (2005). "Branching rules revisited". In: *Operations Research Letters* 33.1, pp. 42–54. DOI: 10.1016/j.orl.2004.04.002.

Adelgren, N. and A. Gupte (2022). "Branch-and-bound for biobjective mixed-integer linear programming". In: *INFORMS Journal on Computing* 34.2, pp. 909–933. DOI: 10.1287/ijoc.2021.1092.

An, D., S. N. Parragh, M. Sinnl, and F. Tricoire (2022). *A matheuristic for tri-objective binary integer programming*. DOI: 10.48550/ARXIV.2205.03386.

Aneja, Y. P. and K. P. K. Nair (1979). "Bicriteria transportation problem". In: *Management Science* 25.1, pp. 73–78. DOI: 10.1287/mnsc.25.1.73.

Bauß, J. and M. Stiglmayr (2023a). *Adaptive multi-objective branch and bound framework*. Git repository. URL: https://git.uni-wuppertal.de/bauss/adaptive-improvements-of-multi-objective-branch-and-bound.

Bauß, J. and M. Stiglmayr (2023b). *Augmented bi-objective branch and bound framework*. Git repository. URL: https://git.uni-wuppertal.de/bauss/augmented-bi-objective-branch-and-bound.

Bauß, J. and M. Stiglmayr (2023c). *GAP and CFLP test instances*. Git repository. URL: https://git.uni-wuppertal.de/bauss/generalized-assignment-problem-test-instances.

Bauß, J. and M. Stiglmayr (2024a). "Adapting branching and queuing for multi-objective branch and bound". In: *Operations Research Proceedings 2023*. Accepted for publictation. Springer. DOI: 10.48550/arXiv.2311.05980.

Bauß, J. and M. Stiglmayr (2024b). "Augmenting bi-objective branch and bound by scalarization-based information". In: *Mathematical Methods of Operations Research*. DOI: 10.1007/s00186-024-00854-3.

Bauß, J., S. N. Parragh, and M. Stiglmayr (2023). "Adaptive improvements of multi-objective branch and bound". *Submitted to EURO Journal on Computational Optimization*. DOI: 10.48550/arXiv.2312.12192.

Bazgan, C., H. Hugot, and D. Vanderpooten (2009). "Solving efficiently the 0–1 multi-objective knapsack problem". In: *Computers & Operations Research* 36.1, pp. 260–279. DOI: 10.1016/j.cor.2007.09.009.

Bellman, R. E. (1957). *Dynamic Programming*. Princeton University Press, p. 339.

Belotti, P., B. Soylu, and M. M. Wiecek (2013). *A branch-and-bound algorithm for biobjective mixed-integer programs*. Tech. rep. URL: http://www.optimization-online.org/DB_FILE/2013/01/3719.pdf.

Belotti, P., B. Soylu, and M. M. Wiecek (2016). "Fathoming rules for biobjective mixed integer linear programs: Review and extensions". In: *Discrete Optimization* 22, pp. 341–363. DOI: 10.1016/j.disopt.2016.09.003.

Benson, H. P. (1998). "An outer approximation algorithm for generating all efficient extreme points in the outcome set of a multiple objective linear programming problem". In: *Journal of Global Optimization* 13.1, pp. 1–24. DOI: 10.1023/A:1008215702611.

Bérubé, J.-F., M. Gendreau, and J.-Y. Potvin (2009). "An exact ε-constraint method for bi-objective combinatorial optimization problems: Application to the traveling salesman problem with profits". In: *European Journal of Operational Research* 194.1, pp. 39–50. DOI: 10.1016/j.ejor.2007.12.014.

Bixby, R. E. (2012). "A brief history of linear and mixed-integer programming computation". In: *Documenta Mathematica*.

Boland, N., H. Charkhgard, and M. Savelsbergh (2015). "A criterion space search algorithm for biobjective integer programming: The balanced box method". In: *INFORMS Journal on Computing* 27.4, pp. 735–754. DOI: 10.1287/ijoc.2015.0657.

Boland, N., H. Charkhgard, and M. Savelsbergh (2017). "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.

Bowman, V. J. (1976). "On the relationship of the Tchebycheff norm and the efficient frontier of multiple-criteria objectives". In: *Lecture Notes in Economics and Mathematical Systems*. Springer Berlin Heidelberg, pp. 76–86. DOI: 10.1007/978-3-642-87563-2_5.

Bökler, F. and P. Mutzel (2015). "Output-sensitive algorithms for enumerating the extreme nondominated points of multiobjective combinatorial optimization problems". In: *Algorithms - ESA 2015*. Springer Berlin Heidelberg, pp. 288–299. DOI: 10.1007/978-3-662-48350-3_25.

Bökler, F., S. N. Parragh, M. Sinnl, and F. Tricoire (2023). "An outer approximation algorithm for multi-objective mixed-integer linear and non-linear programming". In: *Accepted for publication in: Mathematical Methods of Operations Research*.

Chalmet, L., L. Lemonidis, and D. Elzinga (1986). "An algorithm for the bi-criterion integer programming problem". In: *European Journal of Operational Research* 25.2, pp. 292–300. DOI: 10.1016/0377-2217(86)90093-7.

Chankong, V. and Y. Haimes (1983). *Multiobjective Decision Making: Theory and Methodology*. North-Holland series in system science and engineering. North Holland.

Clímaco, J. C. N. and M. M. B. Pascoal (2016). "An approach to determine unsupported non-dominated solutions in bicriteria integer linear programs". In: *INFOR: Information Systems and Operational Research* 54.4, pp. 317–343. DOI: 10.1080/03155986.2016.1214448.

Dächert, K. and K. Klamroth (2014). "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., J. Gorski, and K. Klamroth (2012). "An augmented weighted Tchebycheff method with adaptively chosen parameters for discrete bicriteria optimization problems". In: *Computers & Operations Research* 39.12, pp. 2929–2943. DOI: 10.1016/j.cor.2012.02.021.

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.

De Santis, M., G. Eichfelder, J. Niebling, and S. Rocktäschel (2020). "Solving multiobjective mixed integer convex optimization problems". In: *SIAM Journal on Optimization* 30.4, pp. 3122–3145. DOI: 10.1137/19m1264709.

Dechter, R. and J. Pearl (1985). "Generalized best-first search strategies and the optimality of A*". In: *Journal of the ACM* 32.3, pp. 505–536. DOI: 10.1145/3828.3830.

Dreyfus, S. E. (1977). *The Art and Theory of Dynamic Programming*. Academic Press, p. 284.

Ehrgott, M. (2005). *Multicriteria Optimization*. Springer. DOI: 10.1007/3-540-27659-9.

Ehrgott, M. and X. Gandibleux (2000). "A survey and annotated bibliography of multiobjective combinatorial optimization". In: *OR Spectrum* 22.4, pp. 425–460. DOI: 10.1007/s002910000046.

Ehrgott, M. and X. Gandibleux (2001). "Bounds and bound sets for biobjective combinatorial optimization problems". In: *Lecture Notes in Economics and Mathematical Systems*. Springer Berlin Heidelberg, pp. 241–253. DOI: 10.1007/978-3-642-56680-6_22.

Ehrgott, M. and X. Gandibleux (2007). "Bound sets for biobjective combinatorial optimization problems". In: *Computers & Operations Research* 34.9, pp. 2674–2694. DOI: 10.1016/j.cor.2005.10.003.

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* 151.1, pp. 119–139. DOI: 10.1016/s0377-2217(02)00595-7.

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, pp. 757–778.

Eichfelder, G., P. Kirst, L. Meng, and O. Stein (2021). "A general branch-and-bound framework for continuous global multiobjective optimization". In: *Journal of Global Optimization* 80.1, pp. 195–227. DOI: 10.1007/s10898-020-00984-y.

Forget, N. and S. N. Parragh (2023). "Enhancing branch-and-bound for multiobjective 0-1 programming". In: *INFORMS Journal on Computing*. DOI: 10.1287/ijoc.2022.0299.

Forget, N., S. L. Gadegaard, K. Klamroth, L. R. Nielsen, and A. Przybylski (2022a). "Branch-and-bound and objective branching with three or more objectives". In: *Computers & Operations Research* 148, p. 106012. DOI: 10.1016/j.cor.2022.106012.

Forget, N., S. L. Gadegaard, and L. R. Nielsen (2022b). "Warm-starting lower bound set computations for branch-and-bound algorithms for multi objective integer linear programs". In: *European Journal of Operational Research* 302.3, pp. 909–924.

Fukuda, K. and A. Prodon (1996). "Double description method revisited". In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, pp. 91–111. DOI: 10.1007/3-540-61576-8_77.

Gadegaard, S. L., L. R. Nielsen, and M. Ehrgott (2019). "Bi-objective branch-and-cut algorithms based on LP relaxation and bound sets". In: *INFORMS Journal on Computing* 31.4, pp. 790–804. DOI: 10.1287/ijoc.2018.0846.

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.

Geoffrion, A. M. (1968). "Proper efficiency and the theory of vector maximization". In: *Journal of Mathematical Analysis and Applications* 22.3, pp. 618–630. DOI: 10.1016/0022-247x(68)90201-1.

Haimes, Y., L. Lasdon, and D. Wismer (1971). "On a bicriterion formation of the problems of integrated system identification and system optimization". In: *IEEE Transactions on Systems, Man, and Cybernetics*, pp. 296–297.

Hamacher, H. W., C. R. Pedersen, and S. Ruzika (2007). "Finding representative systems for discrete bicriterion optimization problems". In: *Operations Research Letters* 35.3, pp. 336–344. DOI: 10.1016/j.orl.2006.03.019.

Hamel, A. H., A. Löhne, and B. Rudloff (2013). "Benson type algorithms for linear vector optimization and applications". In: *Journal of Global Optimization* 59.4, pp. 811–836. DOI: 10.1007/s10898-013-0098-2.

Jesus, A. D., L. Paquete, B. Derbel, and A. Liefooghe (2021). "On the design and anytime performance of indicator-based branch and bound for multi-objective combinatorial optimization". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM. DOI: 10.1145/3449639.3459360.

Jorge, J. (2010). "Nouvelles propositions pour la résolution exacte du sac à dos multi-objectif unidimensionnel en variables binaires. (New propositions for the exact solution of the unidimensional multi-criteria knapsack problem with binary variables)." PhD thesis. University of Nantes.

Jozefowiez, N., G. Laporte, and F. Semet (2012). "A generic branch-and-cut algorithm for multiobjective optimization problems: Application to the multilabel traveling salesman problem". In: *INFORMS Journal on Computing* 24.4, pp. 554–564. DOI: 10.1287/ijoc.1110.0476.

Kaliszewski, I. (1987). "A modified weighted tchebycheff metric for multiple objective programming". In: *Computers & Operations Research* 14.4, pp. 315–323. DOI: 10.1016/0305-0548(87)90069-4.

Kellerer, H., U. Pferschy, and D. Pisinger (2004). *Knapsack Problems*. Springer Berlin Heidelberg. DOI: 10.1007/978-3-540-24777-7.

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* 232.3, pp. 479–488. DOI: 10.1016/j.ejor.2013.08.001.

Kirlik, G. and S. Sayın (2015). "Computing the nadir point for multiobjective discrete optimization problems". In: *Journal of Global Optimization* 62.1, pp. 79–99. DOI: 10.1007/s10898-014-0227-6.

Kiziltan, G. and E. Yucaoğlu (1983). "An algorithm for multiobjective zero-one linear programming". In: *Management Science* 29.12, pp. 1444–1453. DOI: 10.1287/mnsc.29.12.1444.

Klamroth, K. and M. M. Wiecek (2000). "Dynamic programming approaches to the multiple criteria knapsack problem". In: *Naval Research Logistics (NRL)* 47.1, pp. 57–76. DOI: 10.1002/(sici)1520-6750(200002)47:1<57::aid-nav4>3.0.co;2-4.

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* 9.4, pp. 378–385. DOI: 10.1016/0377-2217(82)90182-5.

Klötzler, R. (1978). "Multiobjective dynamic programming". In: *Mathematische Operationsforschung und Statistik. Series Optimization* 9.3, pp. 423–426. DOI: 10.1080/02331937808842507.

Kostreva, M. M. and L. C. Lancaster (2008). "Multiple objective dynamic programming". In: *Encyclopedia of Optimization*. Springer US, pp. 2497–2503. DOI: 10.1007/978-0-387-74759-0_430.

Köksalan, M. and B. Lokman (2015). "Finding nadir points in multi-objective integer programs". In: *Journal of Global Optimization* 62.1, pp. 55–77. DOI: 10.1007/s10898-014-0212-0.

Laumanns, M., L. Thiele, and E. Zitzler (2006). "An efficient, adaptive parameter variation scheme for metaheuristics based on the epsilon-constraint method". In: *European Journal of Operational Research* 169.3, pp. 932–942. DOI: 10.1016/j.ejor.2004.08.029.

Leitner, M., I. Ljubić, M. Sinnl, and A. Werner (2016). "ILP heuristics and a new exact method for bi-objective 0/1 ILPs: Application to FTTx-network design". In: *Computers & Operations Research* 72, pp. 128–146. DOI: 10.1016/j.cor.2016.02.006.

Löhne, A. and B. Weißing (2017). "The vector linear program solver Bensolve – notes on theoretical background". In: *European Journal of Operational Research* 260.3, pp. 807–813. DOI: 10.1016/j.ejor.2016.02.039.

Mavrotas, G. and D. Diakoulaki (1998). "A branch and bound algorithm for mixed zero-one multiple objective linear programming". In: *European Journal of Operational Research* 107.3, pp. 530–541. DOI: 10.1016/s0377-2217(97)00077-5.

Mavrotas, G. and D. Diakoulaki (2005). "Multi-criteria branch and bound: A vector maximization algorithm for mixed 0-1 multiple objective linear programming". In: *Applied Mathematics and Computation* 171.1, pp. 53–71. DOI: 10.1016/j.amc.2005.01.038.

Mavrotas, G. (2009). "Effective implementation of the ε-constraint method in multi-objective mathematical programming problems". In: *Applied Mathematics and Computation* 213.2, pp. 455–465. DOI: 10.1016/j.amc.2009.03.037.

Miettinen, K. (1998). *Nonlinear Multiobjective Optimization*. Springer US. DOI: 10.1007/978-1-4615-5563-6.

Morrison, D. R., S. H. Jacobson, J. J. Sauppe, and E. C. Sewell (2016). "Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning". In: *Discrete Optimization* 19, pp. 79–102. DOI: 10.1016/j.disopt.2016.01.005.

Nemhauser, G. and L. Wolsey (1988). *Integer and Combinatorial Optimization*. John Wiley & Sons, Inc. DOI: 10.1002/9781118627372.

Nemhauser, G. L. (1966). *Introduction to Dynamic Programming*. John Wiley & Sons Inc, p. 278.

Özlen, M. and M. Azizoğlu (2009). "Multi-objective integer programming: A general approach for generating all non-dominated solutions". In: *European Journal of Operational Research* 199.1, pp. 25–35. DOI: 10.1016/j.ejor.2008.10.023.

Özpeynirci, Ö. and M. Köksalan (2010). "An exact algorithm for finding extreme supported nondominated points of multiobjective mixed integer programs". In: *Management Science* 56.12, pp. 2302–2315. DOI: 10.1287/mnsc.1100.1248.

Parragh, S. N. and F. Tricoire (2019). "Branch-and-bound for bi-objective integer programming". In: *INFORMS Journal on Computing* 31.4, pp. 805–822. DOI: 10.1287/ijoc.2018.0856.

Przybylski, A. and X. Gandibleux (2017). "Multi-objective branch and bound". In: *European Journal of Operational Research* 260.3, pp. 856–872. DOI: 10.1016/j.ejor.2017.01.032.

Przybylski, A., X. Gandibleux, and M. Ehrgott (2008). "Two phase algorithms for the bi-objective assignment problem". In: *European Journal of Operational Research* 185.2, pp. 509–533. DOI: 10.1016/j.ejor.2006.12.054.

Przybylski, A., X. Gandibleux, and M. Ehrgott (2010a). "A recursive algorithm for finding all nondominated extreme points in the outcome set of a multiobjective integer programme". In: *INFORMS Journal on Computing* 22.3, pp. 371–386. DOI: 10.1287/ijoc.1090.0342.

Przybylski, A., X. Gandibleux, and M. Ehrgott (2010b). "A two phase method for multi-objective integer programming and its application to the assignment problem with three objectives". In: *Discrete Optimization* 7.3, pp. 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].

Schrijver, A. (2003). *Combinatorial Optimization*. Springer, p. 1800.

Serafini, P. (1987). "Some considerations about computational complexity for multi objective combinatorial problems". In: *Recent Advances and Historical Development of Vector Optimization*. Springer Berlin Heidelberg, pp. 222–232. DOI: 10.1007/978-3-642-46618-2_15.

Sourd, F. and O. Spanjaard (2008). "A multiobjective branch-and-bound framework: application to the biobjective spanning tree problem". In: *INFORMS Journal on Computing* 20.3, pp. 472–484. DOI: 10.1287/ijoc.1070.0260.

Steuer, R. E. (1986). *Multiple Criteria Optimization: Theory, Computation and Application*. John Wiley, New York.

Steuer, R. E. and E.-U. Choo (1983). "An interactive weighted Tchebycheff procedure for multiple objective programming". In: *Mathematical Programming* 26.3, pp. 326–344. DOI: 10.1007/BF02591870.

Stidsen, T. and K. A. Andersen (2018). "A hybrid approach for biobjective optimization". In: *Discrete Optimization* 28, pp. 89–114. DOI: 10.1016/j.disopt.2018.02.001.

Stidsen, T., K. A. Andersen, and B. Dammann (2014). "A branch and bound algorithm for a class of biobjective mixed integer programs". In: *Management Science* 60.4, pp. 1009–1032. DOI: 10.1287/mnsc.2013.1802.

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* 6.3, pp. 295–310. DOI: 10.1023/a:1009670112978.

Ulungu, E. 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, pp. 149–156.

Ulungu, E. and J. Teghem (1997). "Solving multi-objective knapsack problem by a branch-and-bound procedure". In: *Multicriteria Analysis*. Springer Berlin Heidelberg, pp. 269–278. DOI: 10.1007/978-3-642-60667-0_26.

Vincent, T., F. Seipp, S. Ruzika, A. Przybylski, and X. Gandibleux (2013). "Multiple objective branch and bound for mixed 0-1 linear programming: Corrections and improvements for the biobjective case". In: *Computers & Operations Research* 40.1, pp. 498–509. DOI: 10.1016/j.cor.2012.08.003.

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, pp. 139–155. DOI: 10.1023/A:1008258310679.

Ziegler, G. M. (1995). *Lectures on Polytopes*. Springer New York. DOI: 10.1007/978-1-4613-8431-1.

Zitzler, E. and L. Thiele (1999). "Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach". In: *IEEE Transactions on Evolutionary Computation* 3.4, pp. 257–271. DOI: 10.1109/4235.797969.