BERGISCHE
UNIVERSITÄT
WUPPERTAL

New Perspectives on Multi-Objective Knapsack Problems

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

vorgelegt von
Britta Schulze

Wuppertal, Januar 2017

Die Dissertation kann wie folgt zitiert werden:

urn:nbn:de:hbz:468-20170427-094229-0
[http://nbn-resolving.de/urn/resolver.pl?urn=urn%3Anbn%3Ade3Ahbz3A468-20170427-094229-0]

Contents

1 Introduction

Imagine the following situation: *You are a decision-maker of a large company and have to decide on the projects that will be realized in the upcoming year. You have a list of possible projects and reports on the corresponding expected benefits and costs. However, the company sets a strict limit on the budget. So, you have to decide carefully which projects you select to maximize the overall benefit while meeting the budget constraint.*

In mathematical optimization this example can be modeled as a knapsack problem: Given a set of items where each item has assigned values of profit and weight (usually positive), the objective is to select a subset of items that maximizes the profit while a predefined maximum weight is not exceeded. Due to its simple structure and frequent occurrence in real-world applications, the knapsack problem has been studied since quite a long time. In 1975, George Bernard Dantzig (1914 – 2005) introduced the name *knapsack problem*, maybe motivated by his example of packing a knapsack or bag for a hike, and, since then, this name is, in general, used by the optimization community.

We extend our example: *The company also limits the available storage area. This additional constraint complicates the decision since the costs and the required storage space of a project are, in general, independent of each other.*

Two constraints are taken into account, thus, the problem is now a bi-dimensional knapsack problem. Actually, multi-dimensional knapsack problems were first motivated by such budget planning scenarios. Additional constraints make the knapsack problem a particularly challenging problem.

Another extension: *The company decides to improve on their sustainability index. So you also have to take into account the impact of each project on this index to maximize the overall improvement. This makes the decision even harder since the rating of profitability and sustainability may be conflicting.*

The problem is now a bi-objective bi-dimensional knapsack problem since we take two objectives into account. The field of multi-objective optimization is concerned with optimization problems with multiple objectives. In general, these objectives are conflicting in the sense that no feasible solution exists that is optimal for each of them. Therefore, compromise solutions have to be found. A compromise solution is a solution that cannot be improved with respect to one objective without reducing at least one other objective function value. In multi-objective optimization these compromise solutions are called *non-dominated*, *efficient* or *(Edgeworth-)Parteo optimal*. Already in the late 19th century, Francis Ysidro Edgeworth (1845 – 1926) and Vilfredo Pareto (1848 – 1923) defined the most common optimality concepts of multi-objective optimization. In many real-world applications, two or more objective functions are needed to model the optimization problem.

Outline of this thesis

In this thesis we consider several extensions of the classical knapsack problem. We treat multi-objective, multi-dimensional and quadratic knapsack problems, and investigate transformations among them to get new insights on properties of the problems and connections between them.

- We present interrelations between supported points, the weight space and concepts from combinatorial geometry. These insights are used to formulate efficient algorithms to compute the set of supported points of multi-objective unconstrained combinatorial optimization problems as well as of multi-objective knapsack problems with positive and negative coefficients. The set of supported points provides a meaningful representation for multi-objective combinatorial optimization problems.
- We analyze the trade-off between constraint satisfaction and objective value by transforming "soft" constraints of multi-dimensional knapsack problems into objective functions. For the resulting multi-objective knapsack problem we present an efficient algorithm that computes the original optimal solution and further efficient solutions that are "close" to it.
- We introduce rectangular knapsack problems as a special case of quadratic knapsack problems. Rectangular knapsack problems constitute a completely new concept for a closed formulation for hypervolume maximization. The optimal solution of the rectangular knapsack problem is a representative solution of a bi-objective knapsack problem.

In our studies we use a large variety of methodological tools, including scalarization methods, combinatorial geometry, dichotomic search, dynamic programming and approximation, among others. All results are interlinked by taking an inherently multi-objective perspective on knapsack problems. The thesis is organized in six chapters, where the current introduction constitutes Chapter 1. In the following we describe the content of the remaining chapters in more detail.

Chapter 2 All notions, definitions and concepts from the literature that will be important in the remainder of this work are presented in this chapter. It is organized in two parts. First, we give a short introduction to multi-objective combinatorial optimization which includes general definitions of multi-objective optimization. In the second part, we present a literature review on single-objective single-dimensional, multi-objective single-dimensional, single-objective multi-dimensional, and multi-objective multi-dimensional knapsack problems, respectively. Structural properties and solution approaches are reviewed. We restrict ourselves to exact and approximate solution approaches.

Chapter 3 We aim for an efficient procedure that computes the set of extreme supported points of two variants of the multi-objective knapsack problem. Extreme supported points

are extreme points of the convex hull of all points that correspond to efficient solutions. While knapsack problems are usually considered solely with positive coefficients, we extend the knapsack model to (intKP) by expanding the domain of objective function coefficients to integer numbers, i. e., by allowing positive and negative coefficients. This yields new interesting theoretical insights and is quite reasonable since it is more convenient to model conflicting objectives as in the case of only positive coefficients. In the first part of the chapter, we relax (intKP) by removing the capacity constraint. The resulting problem is a multi-objective unconstrained combinatorial optimization problem (MUCO). We show that (MUCO) and the corresponding weight space decomposition are interrelated with zonotopes and arrangements of hyperplanes, which are well-known concepts in combinatorial geometry. These interrelations reveal that, for a fixed number of objectives, the number of extreme supported solutions and, therefore, also the number of extreme supported points of (MUCO) is polynomially bounded with respect to the number of items. Furthermore, using the structure of the problem, an efficient algorithm for computing these solutions is formulated. In the second part of the chapter, we review several known algorithms for computing the set of extreme supported points of (intKP). The newly developed algorithm for (MUCO) can be used as a preprocessing for all of these approaches since a certain amount of extreme supported points of (MUCO) remains feasible for (intKP). We present a computational study on tri-objective instances of (MUCO) and of (intKP).

Chapter 4 We consider bi-dimensional knapsack problems where one of the constraints is soft, i. e., a constraint for which the right-hand side is not precisely fixed or uncertain. We reformulate these problems as bi-objective knapsack problems where the soft constraint is relaxed and interpreted as an additional objective function. In this way, a sensitivity analysis for the bi-dimensional knapsack problem can be performed: The trade-off between constraint satisfaction, on the one hand, and the original objective value, on the other hand, can be analyzed. It is shown that a dynamic programming based solution approach for the bi-objective knapsack problem can be adapted in such a way that a representation of the nondominated set is obtained at moderate extra cost. In this context, we are particularly interested in representations of that part of the nondominated set which is in a certain sense close to the constrained optimum in the objective space. We discuss strategies for bound computations and for handling negative cost coefficients, which occur through the transformation. Numerical results comparing the bi-dimensional and bi-objective approaches are presented.

Chapter 5 In this chapter we introduce the cardinality constrained rectangular knapsack problem. It is defined by a quadratic objective function, where the coefficient matrix is the product of two vectors, and a cardinality constraint, i. e., the number of selected items is bounded. We present structural properties of this particular problem and prove upper and lower bounds on the optimal objective function value. These bounds are used to formulate an approximation algorithm. We prove that this algorithm has a polynomial running time with respect to the number of items and guarantees an approximation ratio of 4.5. We also

formulate an improved approximation algorithm and perform computational experiments. The connection between the cardinality constrained rectangular knapsack problem and the cardinality constrained bi-objective knapsack problem is analyzed. We show that the first problem can be used to find a representative solution of the second problem that is optimal for the hypervolume indicator. Further ideas concerning this concept and possible extensions are discussed.

Chapter 6 We summarize the results of this thesis in the last chapter. Ideas for further research are presented at the end of the corresponding Chapters 2, 3 and 4.

Credits

Chapter 2 contains material from a collaborative work with David Willems, Michael Stiglmayr, Stefan Ruzika, Luís Paquete, Kathrin Klamroth, Pascal Halffmann, Carlos Fonseca and José Rui Figueira [see Figueira et al., 2016]. The results on the transformation from a bi-dimensional to a bi-objective knapsack problem in Chapter 4 were obtained by a cooperation with Luís Paquete, Kathrin Klamroth and José Rui Figueira [see Schulze et al., 2017]. Chapter 5 is based on a joint work with David Willems, Michael Stiglmayr, Stefan Ruzika, Luís Paquete and Carlos Fonseca.

Acknowledgement

During my work on this thesis, there happened so many things. Time has passed and it is, unfortunately, hard to remember every person that supported me, especially with the sometimes so valuable little things. So, first of all, to everybody who helped me in some way:

Thank you!

Anyway, there are people who I want to mention:

My special thanks go to my supervisor Kathrin Klamroth: Thank you for giving me this opportunity, for your support, your advice and your positive energy. I am grateful to Luís Paquete for being the "Zweitgutachter" of my thesis, for his encouragement and his useful critiques on this research work. I thank Michael Stiglmayr for spending his time, for his motivation and for his persnicketiness with LaTeX.

I spend a very good time with my former and present colleagues from the *Group of Optimization and Approximation* and from the *Mathewerkstatt* of the university of Wuppertal. I thank Jens Wintermayr, Kirsten Wilshaus, Martin Wagner, Andrea Wagner, Michael Stiglmayr, Teresa Schnepper, Anthony Przybylski, Markus Osenberg, Marco Milano, Renaud Lacour, Kathrin Klamroth, Markus Kaiser, Margareta Heilmann, Anna-Louise Grensing, Kerstin Dächert, Magdalena Boos, Peter Beisel and Katharina Baumann for the pleasant atmosphere and for all the inspiring technical and personal discussions.

My thanks go to the co-authors of two joint publications, to David Willems, Michael Stiglmayr, Stefan Ruzika, Luís Paquete, Kathrin Klamroth, Pascal Halffmann, Carlos Fonseca and José Rui Figueira, for the enjoyable and productive work.

I am thankful for all the journeys that I went on. My thanks go to the friendly hosts, to Margaret Wiecek, Stefan Ruzika, Anthony Przybylski, Luís Paquete, Xavier Gandibleux, Carlos Fonseca, José Rui Figueira and Audrey Cerqueus, and to all fellow travelers for the inspiring and "absolutely fantastic" times in Clemson, Coimbra, Koblenz and Nantes. Thank you to the DAAD for supporting some of these travels (Project-IDs 57128839 and 57211227).

And of course I want to thank my family and friends for supporting me, motivating me and also for sometimes distracting me from work.

List of abbreviations and notation

(MOCO)	multi-objective combinatorial optimization problem
(MUCO), (mo.0c)	multi-objective unconstrained combinatorial optimization problem
(mo.dc)	multi-objective multi-dimensional knapsack problem with m objectives and d constraints
($\text{mo}_{pn\ldots n}$.1c)	multi-objective knapsack problem with positive coefficients in f_1 and negative coefficients in all remaining objective functions
(Qo.1c)	quadratic knapsack problem
(Ro.1c)	rectangular knapsack problem
(_o.Cc)	knapsack problem with cardinality constraint
$\mathbb{R}^n_{\leqq}$	$\{y \in \mathbb{R}^n : y_j \geq 0, \forall j = 1, \ldots, n\}$
$\mathbb{R}^n_{\geq}$	$\{y \in \mathbb{R}^n_{\leqq} : y \neq 0\}$
$\mathbb{R}^n_{>}$	$\{y \in \mathbb{R}^n : y_j > 0, \forall j = 1, \ldots, n\}$
$\text{conv}(X)$	convex hull of the set $X \subseteq \mathbb{R}^n$
2^X	power set of the set $X \subseteq \mathbb{R}^n$
$\mathbf{0}_n$	$(0, \ldots, 0)^\top \in \mathbb{R}^n$
$\mathbf{1}_n$	$(1, \ldots, 1)^\top \in \mathbb{R}^n$
$\mathcal{X} \subseteq 2^{\{0,1\}^n}$	set of feasible solutions
$\mathcal{X}_E$	set of efficient solutions
$\mathcal{X}_{sE}$	set of supported (efficient) solutions
$\mathcal{X}_{eE}$	set of extreme supported (efficient) solutions
$\mathcal{Y} = f(\mathcal{X}) \subseteq \mathbb{R}^m$	set of feasible points
$\mathcal{Y}_N$	set of nondominated points
$\mathcal{Y}_{sN}$	set of supported (nondominated) points
$\mathcal{Y}_{eN}$	set of extreme supported (nondominated) points
$\widetilde{\mathcal{W}^0}$	normalized weight space
$\mathcal{W}^0$	projected weight space
p_i^j	profit of item i in the j-th objective function
$p_i^\bullet$	$(p_i^1, \ldots, p_i^m)^\top$, vector of profits of item i
w_i^k	weight of item i in the k-th constraint
$w_i^\bullet$	$(w_i^1, \ldots, w_i^d)^\top$, vector of weights of item i
W^k	capacity of the k-th constraint
c_{W^k}	constraint slackness of the k-th constraint

Chapter 3

$\mathcal{P}$	polytope
$\mathcal{Z}$	zonotope
$[u, v]$	closed line segment with endpoints $u, v \in \mathbb{R}^m$, $u \neq v$
e_j	j-th unit vector in $\mathbb{R}^m$
$\text{diag}(\beta_1, \ldots, \beta_m)$	diagonal matrix D with $d_{i,j} = 0$ for $i \neq j$ and $d_{i,i} = \beta_i$, for $i, j = 1, \ldots, m$
$h \in \mathbb{R}^m$	hyperplane in $\mathbb{R}^m$
h^+, h^-	positive, negative half-space in $\mathbb{R}^m$ defined by h
$\varphi^{(k)}$	face of dimension k of an arrangement of hyperplanes in $\mathbb{R}^m$ ($k = m$: cell; $k = m-1$: facet; $k = 1$: edge; $k = 0$: vertex)
$\text{Pos}(\lambda)$, $\text{Pos}(\varphi)$	position vector of point y, face φ
$\rho(\lambda_2, \ldots, \lambda_m)$	weighted utility of item i
$\lambda^0_{j,i}$	root weight of item i with respect to f_j

Chapter 4

$\{\bar{s}^1, \ldots, \bar{s}^q\}$	set of extreme supported points sorted in increasing order of $\bar{s}^j_1$
$\bar{s}^{q+1}$	$\bar{s}^q + \binom{1}{-1}$
S_k	stage k of a dynamic programming algorithm
S_0	$\{(0, 0, 0)\}$, initial stage of a dynamic programming algorithm
$s = (s_1, s_2, s_3)$	state of a dynamic programming algorithm
$\text{ext}(s)$	set of feasible extensions of state s
$\text{ext}^j(s)$	set of feasible extensions e of state s with $\bar{s}^j_1 \leq e_1 < \bar{s}^{j+1}_1$
$u(s)$, $u^j(s)$	upper bounds on state s
$\mathcal{C}(f(x_\alpha), f(x_\beta))$	search zone defined by $f(x_\alpha)$ and $f(x_\beta)$

Chapter 5

ρ	approximation ratio
$P := (p_{ij})_{\substack{i=1,\ldots,n \\ j=1,\ldots,n}}$	profit matrix with $p_{ij} = a_i b_j$ for $i, j = 1, \ldots, n$
k	capacity of cardinality constraint
$\overline{\kappa}$	$\lceil k/2 \rceil$
$\underline{\kappa}$	$\lfloor k/2 \rfloor$
$\mathcal{U}, \mathcal{L}$	upper, lower bound on objective function value

2 Preliminaries

The focus of this thesis is on multi-objective combinatorial optimization (MOCO) and, more specifically, mainly on the class of knapsack problems. We assume that the reader is familiar with the basic concepts of linear, combinatorial and multi-objective optimization. As reference work we refer to, for example, the books of Hamacher and Klamroth [2000], Nemhauser and Wolsey [1999], and Ehrgott [2005]. In addition, complexity theory is assumed to be known by the reader. We refer to the book of Garey and Johnson [1979] for the fundamental concepts.

In this chapter we summarize the notions, definitions and concepts that are relevant for the remainder of this work and give a short overview about some of the most common solution approaches. For a more detailed discussion we recommend the surveys of Figueira et al. [2016], Ehrgott and Gandibleux [2000] and Ulungu and Teghem [1994] on MOCO problems and the books of Kellerer et al. [2004] and Martello and Toth [1990] on knapsack problems.

2.1 Multi-objective combinatorial optimization

Given a finite set $A = \{\alpha_1, \dots, \alpha_n\}$ of items α_i, a combinatorial optimization problem searches for an optimal combination of items. To be more precise, the goal is to find a subset of A that is feasible for the problem specific constraints and that optimizes the objective of the problem.

The optimization problem can refer to different fields of application and, so, the items represent different objects. For example, A can represent a set of projects and the optimization problem shall find an optimal subset of projects to realize. Another example is an optimization problem on a graph where the items represent the edges of the graph and an optimal subset of edges has to be chosen. Throughout this thesis, we equivalently use the term *item* for the elements α_i of the set A and for the index i of this element.

Combinatorial optimization problems can be modeled by introducing binary decision variables x_i that indicate for each item i if it is part of the chosen subset ($x_i = 1$) or not ($x_i = 0$). Thus, a subset of A is represented by a vector $x = (x_1, \dots, x_n)^\top \in \{0,1\}^n$. The vector x is called a *solution* of the optimization problem.

The *set of feasible solutions* of a combinatorial optimization problem is denoted by $\mathcal{X}$. A feasible selection is a subset of the power set of A, thus $\mathcal{X} \subseteq 2^{\{0,1\}^n}$. Throughout this thesis we use the same type of restriction: a *capacity constraint*. Each item α_i is associated with a weight coefficient w_i. A selection is feasible if the sum of weights

assigned to selected items does not exceed a given threshold, the *capacity* W, i. e., if

$$\sum_{i=1}^{n} w_i x_i \leq W.$$

One typical objective function in the field of combinatorial optimization is the *sum objective function*. Each item α_i has one associated objective function coefficient p_i. The sum objective function f determines the sum of coefficients assigned to selected items:

$$f(x) = \sum_{i=1}^{n} p_i x_i.$$

The sum objective function is mainly used in the remainder of this thesis.

In multi-objective combinatorial optimization (MOCO) each item α_i is associated with several objective function coefficients p_i^j, $j = 1, \ldots, m$ with $m \geq 2$. In this way, m objective functions $f_j(x)$ are formulated. The (MOCO)-problem can now be modeled as

$$\operatorname{vmax}_{x \in \mathcal{X}} \; f(x) = (f_1(x), \ldots, f_m(x))^\top. \tag{MOCO}$$

The image of the feasible set $\mathcal{X}$ in the objective space is called the *set of feasible points* and is denoted by $\mathcal{Y} := f(\mathcal{X})$. Let $x \in \mathcal{X}$. The solution x is called *efficient* (or Pareto optimal) if there is no other solution $\bar{x} \in \mathcal{X}$ such that

$$f_j(x) \leq f_j(\bar{x}) \;\text{ for all } j = 1, \ldots, m \quad \text{with} \quad f(x) \neq f(\bar{x}).$$

The corresponding point $f(x)$ is called *nondominated*. Let $x, \bar{x} \in \mathcal{X}$. If $f_j(\bar{x}) \leq f_j(x)$ for all $j = 1, \ldots, m$ and $f(\bar{x}) \neq f(x)$, then solution x *dominates* solution $\bar{x}$ and point $f(x)$ *dominates* point $f(\bar{x})$. The *set of efficient solutions* is denoted by $\mathcal{X}_E \subseteq \mathcal{X}$ and the *set of nondominated points* by $\mathcal{Y}_N \subseteq \mathcal{Y}$.

A solution $x \in \mathcal{X}$ is called *weakly efficient*, if there is no other solution $\bar{x} \in \mathcal{X}$ such that

$$f_j(x) < f_j(\bar{x}) \;\text{ for all } j = 1, \ldots, m.$$

Vectors in $\mathbb{R}^m$ can be ordered based on varying definitions since there exists no canonical ordering for $m \geq 2$. The concept of efficiency (or Pareto optimality) is based on the *componentwise order*: Given two points $y_1, y_2 \in \mathbb{R}^m$, we define

$$y_1 \leq y_2 \;:\Leftrightarrow\; y_1^j \leq y_2^j \;\text{ for all } j = 1, \ldots, m, \text{ and } y_1 \neq y_2.$$

The concept of weak efficiency is based on the *strong componentwise order*:

$$y_1 < y_2 \;:\Leftrightarrow\; y_1^j < y_2^j \;\text{ for all } j = 1, \ldots, m.$$

Another order of special interest is the *lexicographic order*. For $y_1, y_2 \in \mathbb{R}^m$:

$$y_1 \leq_{\text{lex}} y_2 \;:\Leftrightarrow\; y_1 = y_2 \text{ or } y_1^j < y_2^j \text{ for } j = \min\{k \,:\, y_1^k \neq y_2^k, k = 1, \ldots, m\}.$$

Let $\mathcal{S}_m$ denote the symmetric group of order m and let $\pi \in \mathcal{S}_m$ denote a permutation of the numbers $1, \ldots, m$. Let $x \in \mathcal{X}$ and let $f_\pi(x) := (f_{\pi(1)}(x), \ldots, f_{\pi(m)}(x))^\top$. The solution x is called *lexicographically optimal* with respect to π if there is no other solution $\bar{x} \in \mathcal{X}$ such that

$$f_\pi(x) \leq_{\text{lex}} f_\pi(\bar{x}), \quad \text{with} \quad f(x) \neq f(\bar{x}).$$

It is easy to see that lexicographically optimal solutions are efficient.

The Minkovski-sum of sets $\mathcal{A}$ and $\mathcal{B}$ in $\mathbb{R}^m$ is defined as $\mathcal{A}+\mathcal{B} = \{a+b \,:\, a \in \mathcal{A}, b \in \mathcal{B}\}$. Let $\mathbb{R}^m_{\geqq} := \{y \in \mathbb{R}^m \,:\, y_j \geq 0,\ \forall j = 1, \ldots, m\}$ denote the non-negative orthant of $\mathbb{R}^m$ and let $\text{conv}(\mathcal{Y}_N)$ denote the convex hull of the nondominated set. The nondominated set of $\text{conv}(\mathcal{Y}_N)$, that is $\{y \in \text{conv}(\mathcal{Y}_N) \,:\, \text{conv}(\mathcal{Y}_N) \cap \{y + \mathbb{R}^m_{\geqq}\} = \{y\}\}$, is called the *nondominated frontier* [cf. Ehrgott, 2005].

The set of nondominated points $\mathcal{Y}_N$ can be classified into two categories: The set of supported nondominated points (short: *supported points*) $\mathcal{Y}_{sN}$, where all points $y \in \mathcal{Y}_{sN}$ are located on the nondominated frontier, and the set of unsupported nondominated points (short: *unsupported points*) $\mathcal{Y}_{uN}$, with $\mathcal{Y}_{uN} = \mathcal{Y}_N \backslash \mathcal{Y}_{sN}$. Furthermore, the set of supported points that are also extreme points of $\text{conv}(\mathcal{Y}_N)$ is called the set of extreme supported nondominated points (short: *extreme supported points*) $\mathcal{Y}_{eN}$. Points that are supported but not extreme are called nonextreme supported nondominated points (short: *nonextreme supported points*). The corresponding efficient solutions are called supported efficient solutions (short: *supported solutions*), unsupported efficient solutions (short: *unsupported solutions*), extreme supported efficient solutions (short: *extreme supported solutions*), and nonextreme supported efficient solutions (short: *nonextreme supported solutions*), respectively. The set of supported solutions is denoted by $\mathcal{X}_{sE}$, the set of unsupported solutions is denoted by $\mathcal{X}_{uE}$, and the set of extreme supported solutions is denoted by $\mathcal{X}_{eE}$.

For (MOCO), the set of feasible points $\mathcal{Y}$ is compact by definition. Therefore, it is possible to define upper and lower bounds on the set of nondominated points. The *ideal point* $y_\mathcal{I} = (y^1_\mathcal{I}, \ldots, y^m_\mathcal{I})^\top$ is defined by the individual maxima of the m objective functions, i. e.,

$$y^j_\mathcal{I} := \max\{y^j \,:\, y \in \mathcal{Y}\} \quad j = 1, \ldots, m.$$

In general, the ideal point $y_\mathcal{I}$ is no element of $\mathcal{Y}$. Otherwise, i. e., if $y_\mathcal{I} \in \mathcal{Y}$, it dominates all other feasible points and $\mathcal{Y}_N = \{y_\mathcal{I}\}$. The ideal point is a tight upper bound on $\mathcal{Y}_N$. The *nadir point* $y_\mathcal{N} = (y^1_\mathcal{N}, \ldots, y^m_\mathcal{N})^\top$ is defined by the minimal components of all nondominated points for the m objective functions, i. e.,

$$y^j_\mathcal{N} := \min\{y^j \,:\, y \in \mathcal{Y}_N\} \quad j = 1, \ldots, m.$$

The nadir point is a tight lower bound on $\mathcal{Y}_N$. The computation of $y_\mathcal{N}$ is a very difficult task in general (for $m \geq 3$) since it asks for an optimization over the set of nondominated points.

There exist several techniques for solving (MOCO). We want to introduce two common scalarization methods: The *weighted sum scalarization* and the *ε-constraint method*.

Scalarization methods are based on the formulation of one or several parametric single-objective optimization problems that replace the original problem. These problems can be solved using appropriate (single-objective) solution methods and the parameters usually allow to compute a (sub-)set of the set of efficient solutions $\mathcal{X}_E$ of the original problem.

Weighted sum scalarization In this method a weighted sum of the objective functions of (MOCO) is built such that a single objective problem is generated. Thereby, the feasible set remains the same, in particular the number of constraints remains equal. Thus, the problem cannot become more difficult than the multi-objective problem. Gaas and Saaty [1955] introduced the weighted sum for linear programming problems with two objectives as the "parametric function". The weighted sum scalarization is formulated as follows:

$$\max_{x \in \mathcal{X}} \sum_{j=1}^{m} \lambda_j \cdot f_j(x) = \langle \lambda, f(x) \rangle \tag{WS(λ)}$$

where the *weights* λ are in $\mathbb{R}^m$. It is well-known [Geoffrion, 1968] that for $\lambda \in \mathbb{R}^m_{\geq} := \{\lambda \in \mathbb{R}^m : \lambda_j \geq 0, j = 1, \ldots, m\}$ every optimal solution of (WS(λ)) is a weakly efficient solution of the initial problem. For (MOCO) the set of feasible points $\mathcal{Y}$ is discrete and finite. Thus, it holds that for $\lambda \in \mathbb{R}^m_{>} := \{\lambda \in \mathbb{R}^m : \lambda_j > 0, j = 1, \ldots, m\}$ every optimal solution of (WS(λ)) is a supported efficient solution of the initial problem. We call $\mathbb{R}^m_{>}$ the *weight space* in the remainder of this work. Furthermore, every supported efficient solution of the initial problem can be found as an optimal solution of (WS(λ)) using appropriate weights $\lambda \in \mathbb{R}^m_{>}$. But no unsupported solution can be generated by the weighted sum method, which is its major drawback.
Trivially, the optimal solution of a single objective optimization problem does not change if the objective function is multiplied by a positive scalar. Hence, the weighted sum objective function can be transformed so that we obtain a convex combination of the objective function values $f_j(x)$. We define the *normalized weight space* $\widetilde{\mathcal{W}}^0$ as the simplex

$$\widetilde{\mathcal{W}}^0 := \left\{ \lambda \in \mathbb{R}^m_{>} : \sum_{j=1}^{m} \lambda_j = 1 \right\}.$$

Since the sum over all weights λ_j is fixed, the dimension of the respective affine subspace is reduced by one to $m-1$. We project the normalized weight space on $\mathbb{R}^{m-1}_{>}$ by setting $\lambda_1 := 1 - \sum_{j=2}^{m} \lambda_j$ and introduce the *projected weight space* $\mathcal{W}^0$:

$$\mathcal{W}^0 := \left\{ (\lambda_2, \ldots, \lambda_m) \in \mathbb{R}^{m-1}_{>} : \sum_{j=2}^{m} \lambda_j < 1 \right\}.$$

For bi-objective problems, Aneja and Nair [1979] formulate an approach for generating all extreme supported points which is based on weighted sum scalarizations. They first compute both lexicographic maxima y_1 and y_2, that are then stored in increasing order with respect to the first objective. Furthermore, a list of "currently" adjacent points is

generated and initialized with the pair (y_1, y_2). Each pair (y_ℓ, y_r) in this list is explored as follows: The weight $\lambda \in \mathcal{W}^0$ is generated that yields equal objective function values in the corresponding (WS(λ)) problem for both points. (WS(λ)) is solved and if, on the one hand, a new supported point is obtained, it is between y_ℓ and y_r and the lists are updated, and if, on the other hand, y_ℓ and y_r are optimal for (WS(λ)), no extreme supported point can be between them. The pair (y_ℓ, y_r) is discarded and the algorithm continues with the next pair until the list is empty. We describe this dichotomic search in more detail in Section 4.3.1 and provide a literature review on multi-objective generalizations in Section 3.4.1.

ε-constraint method In this method one objective function f_k, $k \in \{1, \dots, m\}$, of the (MOCO) problem is selected as objective function of the parametric problem. All other objective functions are transformed into constraints of the problem by introducing a bound on the respective objective function values. Thus, the ε-constraint scalarization is formulated as follows:

$$\begin{array}{lll} \max & f_k(x) & \\ \text{s.t.} & f_j(x) \geq \varepsilon_j & j = 1, \dots, m,\ j \neq k \\ & x \in \mathcal{X} & \end{array} \qquad (\varepsilon\text{-C})$$

where $\varepsilon \in \mathbb{R}^m$. Note that the component ε_k is not used in (ε-C). The ε-constraint method was first introduced in Haimes et al. [1971] [see also Chankong and Haimes, 1983].
It is well-known that, for any $\varepsilon \in \mathbb{R}^m$, every optimal solution of the ε-constraint problem is weakly efficient for the initial problem. It is an efficient solution of the initial problem if it is the unique optimal solution of (ε-C). Furthermore, every efficient solution of the initial problem can be found by the ε-constraint method for any $k \in \{1, \dots, m\}$ by choosing an appropriate vector $\varepsilon \in \mathbb{R}^m$, e. g., for $\bar{x} \in \mathcal{X}_E$, $\varepsilon = f(\bar{x})$ would be an appropriate choice.

In general, (MOCO) problems are not efficiently solvable. There exist two main reasons for that:

1. Most (MOCO) problems are *intractable*, i. e., the size of the set of nondominated points $|\mathcal{Y}_N|$ may grow exponentially in the size of the problem instance n. This means that there exists no polynomial q such that $|\mathcal{Y}_N|$ is bounded by $\mathcal{O}(q(n))$.

2. The computation of unsupported efficient solutions of (MOCO) may be very demanding. Scalarization methods that are designed to also compute unsupported efficient solutions, e. g., the ε-constraint method, often introduce new capacity constraints on objective function values to the optimization model. Hence, the resulting problems are usually $\mathcal{NP}$-hard and not efficiently solvable.

Thus, in practice, the computational effort for computing the whole nondominated set may be too large or, for a decision-maker, the nondominated set itself may be too large to come to a decision. In these cases one has to think about alternative approaches. One

idea is to compute a representative subset of nondominated points. A set $\mathcal{R} \subseteq \mathcal{Y}_N$ is called a *representation* of $\mathcal{Y}_N$. We refer to the survey paper of Faulkenberg and Wiecek [2010] for a detailed introduction to this topic. The set of extreme supported solutions is a representative subset that can be computed rather easy for (MOCO) compared to the computation of the whole nondominated set. Another idea is to compute an approximation of the nondominated set [see, e. g., Papadimitriou and Yannakakis, 2000, Ruzika and Wiecek, 2005, Vanderpooten et al., 2016]. A set $\mathcal{R} \subseteq \mathbb{R}^m$ is called an *approximation* of $\mathcal{Y}_N$ if no point in $\mathcal{R}$ is dominated by another point in $\mathcal{R}$. Of course, a representation is also an approximation and in the literature the term approximation is often used for both.

The quality of an approximation algorithm can be measured by an approximation factor ρ. A quality guarantee is given by approximation schemes. For the following multi-objective definitions see Erlebach et al. [2002].

For $\rho \geq 1$, a solution x is called a *ρ-approximation* of a solution $\bar{x}$ if

$$f_j(x) \geq \frac{f_j(\bar{x})}{\rho}, \quad j = 1, \ldots, m.$$

A set of feasible solutions $X \subseteq \mathcal{X}$ is called *ρ-approximation of $\mathcal{X}_E$* if for all $\bar{x} \in \mathcal{X}_E$ there exists a solution $x \in X$ that is a ρ-approximation for $\bar{x}$. An algorithm B_ρ that has a ρ-approximation of $\mathcal{X}_E$ as output and runs in polynomial time in the size of the input is called a *ρ-approximation algorithm*.

A family of algorithms that contains, for every fixed constant $\varepsilon > 0$, a $(1+\varepsilon)$-approximation algorithm $B_{1+\varepsilon}$ is called a *polynomial time approximation scheme* (PTAS) for $\mathcal{X}_E$. The family of algorithms is called a *fully polynomial time approximation scheme* (FPTAS) for $\mathcal{X}_E$ if the running time of $B_{1+\varepsilon}$ is polynomial in ε^{-1} and in the size of the input.

2.2 Knapsack problems

The *knapsack problem* (1o.1c) is a combinatorial optimization problem. Given a finite set $\{1, \ldots, n\}$ of items i with assigned profit and weight values p_i and w_i, respectively, and a finite capacity W, the 0–1-knapsack problem decides whether or not to include items. The capacity and profit and weight values are all assumed to be positive integer and each item can be included at most once. The goal is to maximize the overall profit of the selected items under the constraint that the overall weight does not exceed the given capacity. Hence, the model consists of a sum objective function and a capacity constraint:

$$\begin{aligned} \max \quad & f(x) = \sum_{i=1}^{n} p_i x_i \\ \text{s. t.} \quad & \sum_{i=1}^{n} w_i x_i \leq W \\ & x_i \in \{0,1\}, \qquad i = 1, \ldots, n. \end{aligned} \tag{1o.1c}$$

In this thesis, we examine knapsack problems with several sum objective functions or with more than one capacity constraint. Therefore, more generally, we define the *multi-objective multi-dimensional knapsack problem* (mo.dc) that consists of m sum objective functions (objectives) and d capacity constraints (dimensions):

$$\begin{aligned} \text{vmax} \quad & f(x) = \big(f_1(x), \ldots, f_m(x)\big) \\ & \qquad = \left(\sum_{i=1}^{n} p_i^1 x_i, \ldots, \sum_{i=1}^{n} p_i^m x_i \right) \\ \text{s.t.} \quad & \sum_{i=1}^{n} w_i^k x_i \leq W^k, \qquad k = 1, \ldots, d \\ & x_i \in \{0,1\}, \qquad i = 1, \ldots, n. \end{aligned} \tag{mo.dc}$$

For $i = 1, \ldots, n$, the variable x_i is set to 1 if item i is included in the knapsack, otherwise x_i is set to 0.

We assume that the number of items n is always larger than the number of objective functions m and larger than the number of constraints d. The coefficient p_i^j indicates the profit of item i in objective function f_j and all profits p_i^j are defined to be non-negative integer values, for $i = 1, \ldots, n$ and for $j = 1, \ldots, m$. We assume that each item i, for $i = 1, \ldots, n$, has a non-zero profit $p_i^j \neq 0$ in at least one objective f_j, $j \in \{1, \ldots, m\}$, i. e., that the *vector of profits*

$$p_i^\bullet = (p_i^1, \ldots, p_i^m)^\top$$

of item i is not equal to the zero vector $\mathbf{0}_m = (0, \ldots, 0)^\top \in \mathbb{R}^m$. Otherwise, i. e., if $p_i^\bullet = \mathbf{0}_m$, item i does not contribute in any objective function and should, therefore, not be selected for the knapsack. Hence, x_i can be preset to 0.

The coefficient w_i^k indicates the weight of item i in the k-th constraint and all weights w_i^k are defined to be non-negative integer values, for $i = 1, \ldots, n$ and for $k = 1, \ldots, d$. We assume that each item i, for $i = 1, \ldots, n$, has a non-zero weight $w_i^k \neq 0$ in at least one constraint, $k \in \{1, \ldots, d\}$, i. e., that the *vector of weights*

$$w_i^\bullet = (w_i^1, \ldots, w_i^d)^\top$$

of item i is not equal to the zero vector $\mathbf{0}_d$. Otherwise, i. e., if $w_i^\bullet = \mathbf{0}_d$, item i does not consume parts of the capacity in any constraint and should, therefore, be selected for the knapsack. Hence, x_i can be preset to 1.

The right-hand side W^k indicates the capacity of the knapsack of the k-th constraint, for $k = 1, \ldots, d$. We assume that $W^k \in \mathbb{Z}$, for $k = 1, \ldots, d$. To avoid trivial solutions, we assume for $k = 1, \ldots, d$ that $\sum_{i=1}^{n} w_i^k > W^k$ and that $w_i^k < W^k$, for $i = 1, \ldots, n$. Hence, W^k has to be non-negative as well. For $k = 1, \ldots, d$, the *slackness* c_{W^k} of a constraint is defined by

$$c_{W^k} \cdot \sum_{i=1}^{n} w_i^k = W^k.$$

Remark 2.1 *Throughout the thesis, we classify the different variants of the knapsack problem by the identifier* (**Pos1***o*.**Pos2***c*)*. The number of sum objective functions is given at Position* **Pos1** *and the number of capacity constraints is given at Position* **Pos2***. For example,* (*mo*.1*c*) *identifies the multi-objective knapsack problem with m objective functions and one constraint and* (2*o*.2*c*) *identifies the bi-objective bi-dimensional knapsack problem. Some of the considered problems have additional characteristics that are reflected by slight adaptions of the classification. These adaptions are declared at the respective introduction of the models.*

In the following we give an overview about structural properties and solution algorithms known from the literature for four categories of knapsack problems. We start with the classical knapsack problem (1o.1c) and continue with the two extensions including several objectives but only one constraint (mo.1c) or allowing several constraints but only one objective (1o.dc). Finally, we present the multi-objective multi-dimensional knapsack problem (mo.dc).

We want to point out that there exists a huge amount of different kinds of variations and extensions of the knapsack problem in the scientific literature. In the book of Kellerer et al. [2004] several interesting types of knapsack problems are presented. So, the basic version (1o.1c) and its multi-objective and multi-dimensional extensions are part of the fascinating class of knapsack problems. In the remainder of this work we also treat quadratic knapsack problems and multi-objective unconstrained combinatorial optimization problems. The latter ones are no knapsack problems by definition but follow the general concept and the above introduced classification by having m sum objective functions but no constraints.

2.2.1 Single-objective single-dimensional knapsack problems

The (single-objective single-dimensional) knapsack problem (1o.1c) is a classical problem in combinatorial optimization. (1o.1c), as its multi-objective and/or multi-dimensional variants, has a wide range of applications, as, for example, in project selection, cargo loading and capital budgeting, and it appears as a frequent subproblem in more complex situations such as, for example, network design. As a consequence, it has been very well studied in the past.

In contrast to its simple structure the knapsack problem is hard to solve. (1o.1c) is $\mathcal{NP}$-hard and, hence, all of its extensions are $\mathcal{NP}$-hard [Garey and Johnson, 1979]. Therefore, it is very unlikely to find polynomial algorithms for this class of problems.

However, due to the large amount of research on (1o.1c) and due to its nice structural properties, the knapsack problem can be solved relatively efficiently. Several solution algorithms have been developed for its solution. Two basic algorithmic concepts for an exact solution of (1o.1c) are:

Dynamic programming (DP) The idea of DP algorithms is to solve a small subproblem of (1o.1c) first and iteratively extend this solution until the problem has been examined

completely. An extensive introduction to DP can be found in Bellman [1957]. DP is a well-established algorithmic technique for solving optimization problems which exhibit Bellman's *Principle of Optimality* [Bellman, 1957], which says that optimal solutions of the overall problem can be easily constructed by extending optimal solutions of smaller subproblems. Given an optimal solution of (1o.1c) which includes item k, an elimination of this item leads to a new knapsack instance with items $\{1, \dots, k-1, k+1, \dots, n\}$ and capacity $W - w_k$. The remaining solution set is obviously an optimal solution for this new instance. Hence, (1o.1c) satisfies the principle of Bellman.
One possible formulation of a dynamic programming algorithm is the following: The solution process is divided into S stages. For the knapsack problem one stage corresponds to the decision of fixing one variable. Thus, the number of stages is equal to the number of variables, $S = n$. Each stage contains at most T different states corresponding to solutions of exactly one subproblem. For the knapsack problem, there is one state for every possible feasible value of the left hand side of the constraint, i. e., for every possible feasible value of $\sum_{i=1}^{n} w_i x_i$. Overall we obtain at most $T = W + 1$ states per stage. An additional, initial stage is defined by the solution that includes no item.
The states in stage k, $k = 1, \dots, n$, can be evaluated through recursive equations applied on the states of stage $(k-1)$ (or on all previously computed states), which retain the feasibility of each solution. As described before, the recursion is based on iteratively fixing one additional variable x_k to 0 or 1, respectively, leading to new partial solutions that extend partial solutions from the predecessor states. Bellman's principle then guarantees that one optimal solution for each state in stage k can be generated by only using the optimal solutions of the states in stage $(k-1)$ (or in all predecessor states, respectively). Therefore, an overall optimal solution can be computed recursively. For the knapsack problem, the optimal solution in state t of stage k can be obtained by comparing the value of state t of stage $(k-1)$ and p_k added to the value of state $t - w_k$ of stage $(k-1)$, if it exists.
For (1o.1c), DP algorithms can be formulated that take $\mathcal{O}(nW)$-time. So, dynamic programming implies pseudo-polynomial running time of solution algorithms for the $\mathcal{NP}$-hard knapsack problem. Furthermore, (1o.1c) can be solved in polynomial time under smooth analysis using the DP approach by Nemhauser and Ullmann [1969] [see Beier and Vöcking, 2003, for more details].

Branch-and-bound (BB) BB algorithms focus on the whole set of feasible solutions, which is divided into smaller subsets during the branching steps. The union of these subsets still has to constitute the whole set of feasible solutions. During the bounding steps, bounds are computed for a subset aiming at excluding it from further consideration. This can be done if the bounds indicate that an optimal solution cannot be an element of the respective subset. Kolesar [1967] presented the first BB algorithm for (1o.1c) and numerous approaches have been presented later on in the literature.
The maybe most intuitive upper and lower bound for (1o.1c) are based on the LP-relaxation of (1o.1c). It is obtained by relaxing the integrality constraint $x_i \in \{0, 1\}$ to $0 \leq x_i \leq 1$.

In contrast to an optimal solution of (1o.1c), an optimal solution of its LP-relaxation can be obtained rather easily [see Dantzig, 1957]: Each item i can be rated by an *efficiency* value e_i, which is simply the profit to weight ratio

$$e_i := \frac{p_i}{w_i}.$$

We assume that the items $1, \ldots, n$ are sorted in non-increasing order of the efficiency values $e_1, \ldots, e_n$. The first $(b-1)$ items, which have the largest efficiency values, are selected, i. e., $x_i^{\mathrm{LP}} := 1$, for $i = 1, \ldots, b-1$, until the first item b would violate the constraint if included entirely, i. e.,

$$\sum_{i=1}^{b-1} w_i \leq W \quad \text{and} \quad \sum_{i=1}^{b-1} w_i + w_b > W.$$

Item b is called the *break item* and is set to $x_b^{\mathrm{LP}} := \left(W - \sum_{i=1}^{b-1} w_i\right)/w_b$. All remaining items are not included, i. e., $x_i^{\mathrm{LP}} := 0$, for $i = b+1, \ldots, n$. The optimal objective function value of the LP-relaxation $f(x^{\mathrm{LP}})$ is an upper bound for (1o.1c).
The integral solution x^B corresponding to the optimal solution of the LP-relaxation, i. e., the solution with $x_i^B := 1$, for $i = 1, \ldots, b-1$, and $x_i^B := 0$, for $i = b, \ldots, n$, is called *break solution*. The break solution is optimal if $\sum_{i=1}^{b-1} w_i = W$, otherwise it can be arbitrarily bad. However, it provides a lower bound on (1o.1c).
In Section 4.3.2 we present another upper bound introduced by Martello and Toth [1977]. It improves the bound obtained by the LP-relaxation by considering the two cases of selecting the break item or not.
Another approach for the computation of upper bounds is the introduction of additional constraints to the LP-formulation. This was first applied by Balas and Zemel [1980]. The constraints have to be formulated such that, on the one hand, they are redundant for (1o.1c), but, on the other hand, x^{LP} becomes infeasible. This approach leads to a smaller optimal LP-solution value that is still an upper bound on (1o.1c). Valid inequalities for the convex hull of the set of feasible solutions $\mathrm{conv}(\mathcal{X})$ can be used to define such constraints.

Balas and Zemel [1980] observed that the optimal solution x^* and the break solution x^B generally vary in only a few variables and that the corresponding efficiencies are all close to the efficiency e_b. It is somehow intuitive that items with a "high" efficiency are almost certainly selected in an optimal solution and that items with a "low" efficiency are almost certainly not selected. Furthermore, it seems natural that the break item defines a "medium" efficiency and that all items with a comparable efficiency are crucial for selecting the optimal subset. The set of these items is called the *core*.

If an optimal solution x^* of (1o.1c) is known, the core can be exactly specified. We assume that the items $1, \ldots, n$ are again sorted in non-increasing order of the efficiency values. We define:

$$\overline{b} := \max\left\{ i \in \{1, \ldots, n\} \,:\, x_i^* = 1 \right\} \quad \text{and}$$

$$\underline{b} := \min\{i \in \{1, \dots, n\} \,:\, x_i^* = 0\}.$$

The core C is given by the set of items $C = \{\underline{b}, \dots, \overline{b}\}$. All items in $H = \{1, \dots, \underline{b}-1\}$ can be categorized has "highly" efficient and are selected in an optimal solution, i. e., $x_i = 1$, for $i \in H$. All items in $\{1, \dots, n\} \setminus C \setminus H = \{\overline{b}+1, \dots, n\}$ can be categorized has "lowly" efficient and are not selected in an optimal solution, i. e., $x_i = 0$, for $i \notin C \cup H$. The problem (1o.1c) can be reduced to the *core problem*:

$$\begin{aligned} \max \quad & \sum_{i \in C} p_i x_i + \sum_{i \in H} p_i \\ \text{s. t.} \quad & \sum_{i \in C} w_i x_i \leq W - \sum_{i \in H} w_i \\ & x_i \in \{0, 1\}, \qquad i \in C. \end{aligned}$$

For many classes of instances the size of the core is only a small fraction of n. Hence, knowing the core considerably simplifies the optimization problem. However, without knowing an optimal solution in advance, which would render an optimization unnecessary, an approximated core of sufficient size has to be predefined [see, e. g., Martello and Toth, 1988] or a predefined core has to be expanded during the solution algorithm if necessary [see, e. g., Pisinger, 1995].

One of the most efficient algorithms in practice for (1o.1c) was introduced by Martello et al. [1999]. It is called COMBO since it combines several solution concepts. Its general structure is based on the MINKNAP algorithm of Pisinger [1997]. MINKNAP is an expanding core algorithm based on dynamic programming that enumerates the smallest symmetrical core $\{b - \delta, \dots, b + \delta\}$ around the break item during the algorithm. Thus, the computational effort is reduced by keeping the core as small as possible. Furthermore, the effort for sorting items is also kept small. The break item is found by a partial sorting and an upper bound test, with a bound that can be obtained at low cost, is applied to an item to possibly fathom the item before it is sorted. Furthermore, another upper bound test, with a strong upper bound, is applied to sorted items to possibly fathom them before adding them to the core. As soon as an item is included in the core, a recursion of the dynamic programming is executed. COMBO measures the hardness of the considered problem by counting the number of states of the DP. It starts as the MINKNAP algorithm. If the number of states exceeds given thresholds, additional techniques are applied to tighten the upper and lower bounds. COMBO and MINKNAP are very fast in practice. In our computational studies we use MINKNAP to solve (1o.1c).

Several further algorithms have been developed in the past for the exact solution of (1o.1c) and also for computing approximate solutions. An introduction to all of these ideas is beyond the scope of this thesis. Therefore, we restrict ourselves to the approaches that are applied extended or generalized in the reminder of this work. We, again, refer to the book of Kellerer et al. [2004] for further reading.

2.2.2 Multi-objective single-dimensional knapsack problems

The *multi-objective (single-dimensional) knapsack problem* (mo.1c) can be defined as:

$$\begin{aligned} \text{vmax} \quad & f(x) = \big(f_1(x), \dots, f_m(x)\big) \\ & \quad = \Big(\sum_{i=1}^{n} p_i^1 x_i, \dots, \sum_{i=1}^{n} p_i^m x_i \Big) \\ \text{s. t.} \quad & \sum_{i=1}^{n} w_i x_i \leq W \\ & x_i \in \{0,1\}, \qquad i = 1, \dots, n. \end{aligned} \tag{mo.1c}$$

The problem (mo.1c) is the multi-objective extension of (1o.1c) and, thus, inherits its difficulty. Furthermore, (mo.1c) is, in general, intractable. However, exact solution methods developed for the single-objective case can be adapted to the multi-objective problem. Furthermore, the multi-objective structure motivates a new solution concept organized in two phases.

Dynamic programming (DP) Bellman's principle of optimality can be extended to multi-objective optimization problems with additional objective functions $f^2, \dots, f^m$, see Brown and Strauch [1965]. In this case, instead of one solution/state, there may be several efficient solutions/states for every possible feasible value of the left hand side of the constraint, i. e., for every possible feasible value of $\sum_{i=1}^{n} w_i x_i$. Dominated solutions can be pruned during the process, see Figure 2.1 for an illustration. We give a more detailed introduction to DP for (2o.1c) in Section 4.2.
Captivo et al. [2003] apply a DP algorithm on (2o.1c) by transforming it into a bi-objective shortest path problem over an acyclic network. The DP is based on a labeling algorithm for multi-objective shortest path problems. Figueira et al. [2013] introduce three variants of DP approaches that make use of new introduced upper and lower bounds. These are based on the computation of extreme supported solutions, on the convex relaxation of (2o.1c) that is solved by a bi-objective simplex algorithm, and on the computation of an upper bound set for every state. Rong and Figueira [2013, 2014] present several DP approaches for (2o.1c) that prune states of the DP process using different techniques based on core approximations, upper bound sets and structural properties of DP applied on (2o.1c).
For (mo.1c) and other versions of multi-objective knapsack problems Klamroth and Wiecek [2000] give a detailed study of different dynamic programming approaches. Bazgan et al. [2009b] introduce a DP algorithm that uses several complementary dominance relations to prune states during the process that cannot lead to nondominated points of (mo.1c). Figueira et al. [2010] present an algorithm that is based on solving the multiple objective shortest path problem on an underlying network, cf. Captivo et al. [2003]. The authors introduce different network models and algorithms for the respective construction.

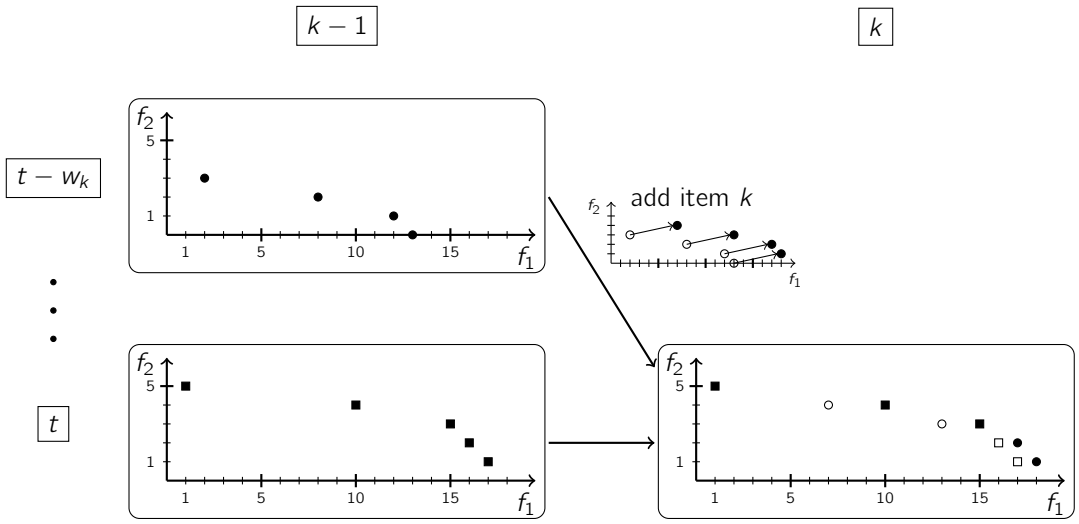


Figure 2.1: For an instance of (2o.1c), this figure illustrates the transformation from states in stage $(k-1)$ to states in stage k in a dynamic programming algorithm: The efficient solutions in stage k, with $\sum_{i=1}^{k} w_i x_i = t$, are obtained from those in stage $k-1$, with $\sum_{i=1}^{k-1} w_i x_i = t - w_k$ and $\sum_{i=1}^{k-1} w_i x_i = t$. The symbols • and ▪ show the nondominated points corresponding to the respective states. The symbols ◦ and ▫ show dominated points in stage k that are filtered out.

Two phase method (2P) Visée et al. [1998] observed in numerical tests that, while the number of efficient solutions for instances of (2o.1c) grows exponentially with the number of items, the number of supported solutions often grows only linearly. Furthermore, applying the weighted sum scalarization, the bi-objective problem reduces to the single-objective knapsack problem (1o.1c) and, thus, supported solutions are rather easy to obtain. Hence, the supported solutions can be computed in a first phase and a DP or BB approach can be applied to compute the unsupported solutions in a second phase. During the second phase the information obtained in the first phase can be used to improve the process. This concept is called the *two-phase approach*.

Visée et al. [1998] present a "breadth first" and a "depth first" BB algorithm for the second phase of (2P) using the upper bound of Martello and Toth [see Martello and Toth, 1990] and several fathoming criteria based on the lower bounds that can be defined by the supported solutions.

Delort and Spanjaard [2010, 2013] use a DP approach in the second phase where one DP procedure is run for each pair of adjacent supported points (y_ℓ, y_r). This pair defines a triangle in the objective space where unsupported points between y_ℓ and y_r can be, see Figure 2.2 for illustration. For each state of the DP process an upper bound set is computed. The state can be pruned if it does not intersect with the triangle defined by y_ℓ and y_r. This pruning strategy is quite successful in practice since each triangle is analyzed individually.

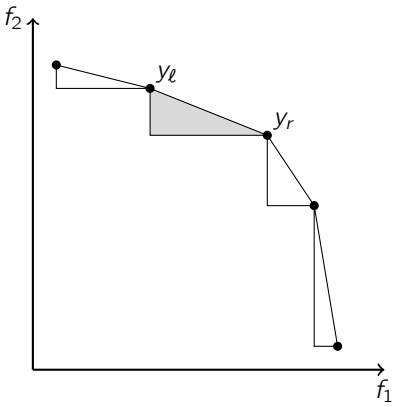


Figure 2.2: Supported points in the objective space. Each pair of adjacent supported points (y_ℓ, y_r) defines a triangle where unsupported points between y_ℓ and y_r can be.

Further approaches Ulungu and Teghem [1997] present a branch-and-bound algorithm that is a bi-objective version of the method of Martello and Toth for (1o.1c) [see Martello and Toth, 1990].
The core concept of the single-objective problem (1o.1c) is based on the definition of the efficiencies e_i that indicate the profit to weight ratios for items $i = 1, \ldots, n$. Including m objective functions, m efficiency values can be defined for each item of (mo.1c). Since the objective functions are, in general, conflicting, these values can differ very much from each other. This makes it complicated to define a core for (mo.1c). However, Gomes da Silva et al. [2008] present a core concept for (2o.1c) assigning a core to each efficient solution. They show that also for the bi-objective version the core contains only few variables and formulate an exact and an approximate method for (2o.1c).
The weighted sum scalarization is generalized to find unsupported nondominated points of (mo.1c) by Gomes da Silva and Clímaco [2013]. The authors introduce perturbation terms for each item in each objective function. These perturbations are used to change the shape of the set of feasible points such that unsupported nondominated points are shifted to the nondominated frontier of the perturbed problem and can be found using the weighted sum scalarization. They show that all unsupported nondominated points can be generated by this approach. However, it is unclear how to define good perturbations and how to determine that the complete set $\mathcal{Y}_N$ has been found to stop the algorithm.

Safer and Orlin [1995] prove the existence of an FPTAS for (mo.1c). Erlebach et al. [2002] and Bazgan et al. [2009a] formulate FPTAS for (mo.1c) based on dynamic programming. Erlebach et al. [2002] point out that it is important to approximate every reachable profit value and not only the optimal solution values. The authors partition the objective space in their approach to prove an approximation guarantee. Bazgan et al. [2009a] use several dominance relations to discard states during the DP process [see also the work on exact algorithms in Bazgan et al., 2009b].

2.2.3 Single-objective multi-dimensional knapsack problems

The *(single-objective) multi-dimensional knapsack problem* (1o.dc) can be defined as:

$$\begin{aligned} \max \quad & f(x) = \sum_{i=1}^{n} p_i x_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} w_i^k x_i \leq W^k, \qquad k = 1, \ldots, d \\ & x_i \in \{0,1\}, \qquad i = 1, \ldots, n. \end{aligned} \tag{1o.dc}$$

(1o.dc) is the multi-dimensional extension of (1o.1c) and, thus, inherits its difficulty. The multi-dimensional knapsack problem was first mentioned in the economical context of rationing capital [Lorie and Savage, 1955]. Markowitz and Manne [1957] introduce formulations of several discrete programming problems. They consider multi-dimensional knapsack problems among others and present a general solution approach which can be adjusted to different discrete problem structures.

One technique that is frequently used to compute upper bounds for (1o.dc) is the *surrogate relaxation*. The problem is relaxed by merging two or several of the constraints. We assume that we want to merge the first $\bar{d}$ constraints. As for the weighted sum scalarization, this is done by a linear combination using non-negative multipliers μ_k, $k = 1, \ldots, \bar{d}$. The *surrogate relaxed problem* (SR(μ)) of (1o.dc) is defined as [see Glover, 1965] :

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} p_i x_i \\ \text{s.t.} \quad & \sum_{k=1}^{\bar{d}} \mu_k \sum_{i=1}^{n} w_i^k x_i \leq \sum_{k=1}^{\bar{d}} \mu_k W^k \\ & \sum_{i=1}^{n} w_i^k x_i \leq W^k, \qquad k = \bar{d}+1, \ldots, d \\ & x_i \in \{0,1\}, \qquad i = 1, \ldots, n. \end{aligned} \tag{SR(μ)}$$

All feasible solutions of (1o.dc) are also feasible for (SR(μ)), thus, for all non-negative multipliers μ, the optimal solution of (SR(μ)) is an upper bound on (1o.dc). The "best" choice of multiplier μ is defined by the smallest objective function value of (SR(μ)) that can be achieved:

$$\min_{\mu \geq 0} SR(\mu).$$

This problem is called the *surrogate dual problem* [see Glover, 1975].

While there are several heuristic approaches for solving bi- or multi-dimensional knapsack problems, there are rather few exact algorithms. For a review, we refer to Fréville [2004] and to Puchinger et al. [2010].

Dynamic Programming (DP) DP algorithms for (1o.1c) can be generalized also to multi-dimensional problems. Weingartner and Ness [1967] and Nemhauser and Ullmann [1969] suggest dynamic programming as solution method. The number of stages is still equal to the number of variables, i. e., $S = n$. Since the d constraints with their respective capacities W^k have to be taken into account, each stage contains at most $\prod_{k=1}^{d}(W^k + 1)$ states. Hence, overall we obtain $\mathcal{O}(n(W_{\max})^d)$ states, where $W_{\max} = \max\{W^k : k = 1, \ldots, d\}$. Thus, DP algorithms for (1o.dc) are generally much less efficient in practice than DP algorithms for (1o.1c), even in the bi-dimensional case. Weingartner and Ness [1967] also present a "dual" approach where all items are selected and the DP decides about excluding items or not. During the process items are excluded until feasible solutions are obtained.

Branch-and-bound (BB) A first BB algorithm for (1o.dc) was given by Thesen [1975]. The algorithm is formulated in a straightforward way and the authors concentrate on the underlying data structure to save core memory space. Shih [1979] compute upper bounds for the BB approach in the following way: They define d instances of (1o.1c) including only the k-th constraint, $k = 1, \ldots, d$. The optimal solution values of the corresponding LP-relaxations are compared and the minimum value is used as an upper bound. Gavish and Pirkul [1985] present LP-, Lagrangian-, and surrogate relaxation for (1o.dc) and a combination of the last two. They analyze the theoretical relation between these relaxations and present extensive computational studies to compare the correspondingly generated bounds. They also introduce concepts to reduce the problem size before and during the BB process. Fréville and Plateau [1996] concentrate on the bi-dimensional case. In a preprocessing phase they apply a problem reduction scheme using, among other strategies, the surrogate dual. Martello and Toth [2003] also study BB concepts for (1o.dc). They present improved techniques for computing optimal Lagrangian and surrogate multipliers. Upper bounds, heuristic approaches and a reduction procedure are combined to formulate an exact BB algorithm.

Further Approaches Boyer et al. [2010] combine the two above methods. They use a dynamic programming heuristic that is based on surrogate relaxation and a branch-and-bound procedure. Boussier et al. [2010] also apply different solution approaches in a combined multi-level search strategy. The items are sorted in decreasing order w. r. t. the reduced costs of the non-basic variables in the corresponding LP-relaxations. The authors use Resolution Search, BB, and a simple Depth First Search (DFS) enumeration, depending on the level of the current branch. Mansini and Speranza [2012] present a core algorithm for the multi-dimensional KP. They split the problem into subproblems with fewer variables and apply a variable fixing algorithm. This procedure is terminated as soon as the number of non-fixed variables drops below a predefined threshold. The resulting problems are named restricted core problems. They are solved by partitioning the solution space into subspaces with a given number of included items and examining these subspaces using a BB procedure.

Gens and Levner [1979] show that there exists no FPTAS for (1o.2c) unless $\mathcal{P}$=$\mathcal{NP}$. Therefore, a PTAS is the best approximation result one may hope for. Frieze and Clarke [1984] present a PTAS which is based on the computation of basic feasible solutions of the LP-relaxation of (1o.dc). An improved version of this PTAS is given by Caprara et al. [2000].

2.2.4 Multi-objective multi-dimensional knapsack problems

The multi-objective multi-dimensional knapsack problem inherits the difficulties of (mo.1c) and (1o.dc). Thus, most publications on (mo.dc) are concerned with heuristic methods. Recently, Lust and Teghem [2012] give a review on existing approaches for (mo.dc).

Branch-and-bound (BB) Florios et al. [2010] extend a multi-objective BB procedure to the multi-dimensional case. They include several branching heuristics and use the ideal point for fathoming branches. Cerqueus [2015] introduces a branch-and-cut method for (2o.2c) that is based on comprehensive studies on all included concepts. The author presents a computational study on the quality of branching strategies from the literature and introduces new upper bounds that are based on surrogate relaxation [see also Cerqueus et al., 2015]. Furthermore, a dynamic branching strategy and valid inequalities are shown.

Further Approaches Laumanns et al. [2006] present an ε-constraint method to solve multi-objective optimization problems. They present an adaptive scheme to generate new ε-values during the solution process. Mavrotas et al. [2009] and Mavrotas et al. [2011] adapt the core concept to solve (2o.dc). The extreme supported points of the LP-relaxation are computed to define appropriate weight vectors λ of the weighted sum scalarization for each of these points. One set of core variables is defined for each point and a subproblem of (2o.dc), only including the core variables, is solved by a multi-objective BB algorithm. At the end, all "local" sets of nondominated points have to be merged to obtain the nondominated set of (2o.dc).

Since (mo.dc) includes (1o.dc) as a special case, unless $\mathcal{P}$=$\mathcal{NP}$, also for (mo.dc) there exists no FPTAS. Erlebach et al. [2002] were the first to present a PTAS for the multi-objective multi-dimensional knapsack problem. The authors define subproblems, containing only one of the objective functions, to obtain upper bounds. They also define lower bounds and use them to define a subspace of the objective space in which an approximate solution should be found. This is done by solving an LP-relaxation of the problem. The authors also provide ideas for a more general version where some of the objectives are to be minimized.

3 Computation of extreme supported points

The computation of the set of supported points in a first phase followed by the computation of all efficient points using the precomputed information in a second phase is a successful concept for (MOCO) problems. As mentioned before, Visée et al. [1998] were motivated by the results from numerical experiments on (2o.1c) to introduce the *two phase method*. The authors observed that the number of supported points usually grows only linearly with the number of items whereas the number of unsupported points grows exponentially. There are examples with an exponential number of supported points, see, for example, Gomes da Silva et al. [2004] and Ehrgott [2005]. It is an open question whether the number of extreme supported points is bounded which we settle here for one class of (MOCO) problems.

There are more reasons why extreme supported points play a central role in multi-objective combinatorial optimization as, for example, that they provide information on achievable ranges of objective values and, in this way, support the decision making process and that they are "maximal" in the sense that they all lie on the convex hull of feasible points and, thus, define a in the same sense "maximal" representation of $\mathcal{Y}_N$.

In this chapter we present concepts and algorithms for an efficient procedure for the computation of extreme supported points for two classes of (MOCO) problems. We start our research with multi-objective unconstrained combinatorial optimization problems (MUCO). These are (MOCO) problems that consist of m sum objective functions and no additional constraints. We interpret (MUCO) as a relaxed version of the multi-objective knapsack problem and introduce our classification for knapsack problems also for (MUCO) denoting it by (mo.0c).

For knapsack problems, the profit coefficients are, in general, assumed to be non-negative. For (mo.0c) this assumption is not reasonable. Having only non-negative profit values, the maximization objectives tend to select all items. This choice is feasible since no constraint limits the selection and, thus, the problem has one unique optimal solution corresponding to the selection of all items. We assume that the objective functions are conflicting which is realized by also allowing negative coefficients. We make this assumption for (mo.0c) as well as for (mo.1c). We show that this allows a new and more general perspective on (mo.1c).

Remark 3.1 *Throughout this chapter we assume that*

$$p_i^j \in \mathbb{Z}, \quad i = 1, \ldots, n,\ j = 1, \ldots, m$$

for (mo.0c) *and for* (mo.1c).

As a consequence, it would be consistent to extend also the weight coefficients w_i, for $i = 1, \ldots, n$, of (mo.1c) to negative values. However, without loss of generality we assume that $w_i \geq 0$, for $i = 1, \ldots, n$. If this is not satisfied, i. e., if $w_i < 0$, for some $i = 1, \ldots, n$, this means that the capacity of the knapsack grows if the item is selected. We can then interpret the variable in the way that the item is included and we decide about excluding it or not. We obtain an equivalent problem with $w_i \geq 0$, for all $i = 1, \ldots, n$, by substituting the variable x_i by its complement $\bar{x}_i$ which is set to 1 if item i is excluded and set to 0 if it is not excluded. Accordingly, all coefficients have to be replaced by their negative value, i. e., $\bar{p}_i^j = -p_i^j$, for $j = 1, \ldots, m$, and $\overline{w}_i = -w_i > 0$. The capacity has to be increased by the absolute value of the weight, i. e., $\overline{W} = W - w_i > W$.

Ehrgott [2005] proves that (mo.0c) is intractable. Gorski et al. [2011] analyze the connectedness of efficient solutions in multi-objective combinatorial optimization problems. They define that two efficient solutions x and x' of (mo.0c) are called *adjacent* iff x and x' differ in exactly one component, i. e., if $\sum_{i=1}^{n} |x_i - x'_i| = 1$. The authors show that the corresponding adjacency graph is non-connected in general whereas it always contains a connected subgraph, which is the subgraph of supported efficient solutions. Our results confirm this second result and, furthermore, prove a polynomial bound on the number of extreme supported solutions for (mo.0c). Liefooghe et al. [2013] present an experimental analysis on the connectedness of (2o.0c). They call two efficient solutions adjacent if they differ in exactly one component or if one solution can be obtained from the other by exchanging two items. All instances of their experimental study are connected with respect to this definition of adjacency.

We prove the polynomial bound on the number of extreme supported solutions for (mo.0c) using concepts from combinatorial geometry, namely *arrangements of hyperplanes* and *zonotopes*. Furthermore, we present conditions for the occurrence of nonextreme supported solutions. Seipp [2013] presents a polynomial bound on the number of extreme supported points for multi-objective minimum spanning tree problems. The author proves his result also by using arrangements of hyperplanes. Aissi et al. [2015] study the number of supported nondominated cuts in graphs and hypergraphs with multiple edge cost functions. They prove a polynomial bound with respect to the number of nodes and edges. However, their result is based on bounds on the number of approximate global minimum cuts and not on arrangements of hyperplanes.

The results on (mo.0c) can be applied to improve existing algorithms for computing the set of extreme supported points of (mo.1c) in the generalized form, i. e., assuming the profit coefficients to be positive or negative integers. The extreme supported points of the unconstrained (relaxed) version of (mo.1c) are generated and all points that remain feasible for (mo.1c) are used as an initial set.

This chapter is structured as follows: In Section 3.1 we introduce definitions that are necessary for the remainder of this chapter. We present the interrelations between the multi-objective unconstrained combinatorial optimization problem and concepts from combinatorial geometry in Section 3.2. These interrelations are used to prove the polynomial bound on the number of extreme supported solutions.

In Section 3.3 we analyze the properties of (mo.1c) and the relation to the previously introduced concepts in more detail. To do so we start with the bi-objective case and introduce a first solution algorithm. We generalize the problem specific observations to three and more objectives and present a case study on tri-objective unconstrained combinatorial optimization problems with positive coefficients in the first and negative coefficients in the second and third objective function. This case study leads to the formulation of an adapted algorithm. Corresponding computational results are presented. Concluding, we discuss generalizations of the algorithmic concept to problems with arbitrary coefficients and with an arbitrary number of objectives. We also present how all extreme points of the convex hull of the set of feasible points can be computed.

An efficient computation of the set of extreme supported points for (mo.1c) is presented in Section 3.4. At first we survey existing literature on this topic and explain how the previous results can be used to improve these concepts. In more detail we, again, present a case study on tri-objective knapsack problems with positive coefficients in the first and negative coefficients in the second and third objective function. This induces structural properties of that we take advantage in a solution algorithm. A description of the approach and computational results are given. We conclude with a summary and further ideas in Section 3.5.

3.1 Definitions

In this section, we give a short introduction to multi-objective unconstrained combinatorial optimization problems and present the idea of weight space decomposition. Furthermore, the studies of this chapter use several concepts from combinatorial geometry. In the following, we recall the required definitions.

3.1.1 Multi-objective unconstrained combinatorial optimization problems

The *multi-objective unconstrained combinatorial optimization problem* ((MUCO) or, in our notation, (mo.0c)) has, similar to (mo.1c), m sum objective functions, but does not have a capacity constraint:

$$\begin{aligned} \text{vmax} \quad & f(x) = \left(\sum_{i=1}^{n} p_i^1 x_i, \ldots, \sum_{i=1}^{n} p_i^m x_i \right) \\ \text{s.t.} \quad & x_i \in \{0,1\}, \qquad i = 1, \ldots, n. \end{aligned} \tag{mo.0c}$$

We assume that all coefficients p_i^j are integers, for $i = 1, \ldots, n$ and for $j = 1, \ldots, m$. For consistency, also for (mo.0c) we call p_i^j the profit of item i in objective function j.

Ehrgott [2005] prove that (mo.0c) is $\mathcal{NP}$-complete, even for $m = 2$. However, the single-objective unconstrained combinatorial optimization problem (1o.0c) can be solved explicitly.

Theorem 3.2 *All solutions x with*

$$x_i \begin{cases} = 0 & \text{if } p_i^1 < 0 \\ \in \{0,1\} & \text{if } p_i^1 = 0 \\ = 1 & \text{if } p_i^1 > 0 \end{cases}$$

for $i = 1, \ldots, n$, are optimal for (1o.0c) and there exist no further optimal solutions.

Proof. Items with positive coefficients p_i^1 increase the objective function value of (1o.0c) and, therefore, they are included in all optimal solutions. Items with negative coefficients p_i^1 decrease the objective function value and are not included in any optimal solution. Items with coefficients $p_i^1 = 0$ do not contribute to the objective function value. Those items lead to alternative optimal solutions because including or not including them results in the same objective function value. □

One important observation is that the decision on one item can be made independently of the decisions on all other items. Since there is no constraint, the variables are not interlinked. Certainly, the multi-objective version (mo.0c) is more complicated, but this independence is preserved. As long as all coefficients of an item i, $i = 1, \ldots, n$, have equal signs for all objective functions, Theorem 3.2 can still be applied. However, in general, the objective functions are conflicting and, therefore, the signs of the coefficients differ.

3.1.2 Polyhedra and zonotopes

A set $\mathcal{P} \in \mathbb{R}^m$ is called *polyhedron* if it is the intersection of finitely many half-spaces. If $\mathcal{P}$ is bounded, it is called *polytope*. It can be shown that the convex hull $\text{conv}(\mathcal{A})$ of a finite set of points $\mathcal{A}$ in $\mathbb{R}^m$ is a polytope. Let a be the number of affinely independent points in $\mathcal{P}$. The dimension $\dim(\mathcal{P})$ of $\mathcal{P}$ is defined as $\dim(\mathcal{P}) = a - 1$.

For a polyhedron $\mathcal{P} \subseteq \mathbb{R}^m$, an inequality $\lambda y \leqq \lambda_0$ is called a *valid inequality* if it is satisfied for all $y \in \mathcal{P}$. A subset $\varphi \subseteq \mathcal{P}$ is called a *face* of $\mathcal{P}$ if $\varphi = \{y \in \mathcal{P} : \lambda y = \lambda_0\}$ for some valid inequality $\lambda y \leqq \lambda_0$ of $\mathcal{P}$. It can be shown that a face of a polyhedron is again a polyhedron. A face is called k-face if it is a polyhedron of dimension k and $(m-1)$-faces are called facets [definitions and results following Wolsey, 1998].

For all binary optimization problems, particularly for (mo.0c) and (mo.dc), the convex hull $\text{conv}(\mathcal{Y})$ of the set of feasible points is a polytope since all variables and, therefore, the feasible points are bounded. Faces of $\text{conv}(\mathcal{Y})$ are called nondominated if they are part of the nondominated frontier.

A set $\mathcal{Z} \subset \mathbb{R}^m$ is called a *zonotope* if it is the Minkovski-sum of a finite number of closed line segments $[u_i, v_i] = \{y \in \mathbb{R}^m : y = u_i + \mu \cdot (v_i - u_i), \mu \in [0,1]\}$, with vectors $u_i,\ v_i \in \mathbb{R}^m$, $u_i \neq v_i$, for $i = 1, \ldots, n$. Zonotopes $\mathcal{Z}$ are polytopes.

The center of a zonotope can be generated by summing over the midpoints $\bar{y}_i$ of each line segment, where $\bar{y}_i = u_i + \frac{1}{2} \cdot (v_i - u_i)$. Zonotopes are centrally symmetric, which can easily be seen by translating each line segment such that the origin is its midpoint [definitions and results following Edelsbrunner, 1987].

3.1.3 Weight space decomposition

As introduced in Section 2.1, the weighted sum scalarization can be applied to find the extreme supported solutions of a (MOCO) problem. Benson and Sun [2000] show for multi-objective linear programming problems that there exists a one-to-one correspondence between supported solutions and subsets of the projected weight space $\mathcal{W}^0$. The weight space decomposition gives the weight vectors that lead to each efficient basic solution. Przybylski et al. [2010a] analyze this interrelation for multi-objective integer programming problems with m objective functions: For a supported point $y \in \mathcal{Y}_{sN}$, we define

$$\mathcal{W}^0(y) := \left\{\lambda \in \mathcal{W}^0 \,:\, \langle \lambda, y\rangle = \min\{\langle \lambda, \bar{y}\rangle \,:\, \bar{y} \in \mathcal{Y}_{eN}\}\right\}$$

which is the subset of weights λ in the projected weight space that define weighted sum problems for which y is optimal. Note that a weight λ can be assigned to several sets $\mathcal{W}^0(y)$ if the corresponding points y realize the minimum value of $\{\langle \lambda, \bar{y}\rangle \,:\, \bar{y} \in \mathcal{Y}_{eN}\}$.

The set $\mathcal{W}^0(y)$ is a polytope and y is an extreme supported point if and only if $\dim(\mathcal{W}^0(y)) = m - 1$. Furthermore, for two supported points y and $\bar{y}$ either the intersection of $\mathcal{W}^0(y)$ and $\mathcal{W}^0(\bar{y})$ is empty or it is the common face of maximal dimension. The sets $\mathcal{W}^0(y)$, for $y \in \mathcal{Y}_{eN}$, define a decomposition of $\mathcal{W}^0$, i. e.,

$$\mathcal{W}^0 = \bigcup_{y \in \mathcal{Y}_{eN}} \mathcal{W}^0(y).$$

All definitions and statements can be done analogously for the weight space $\mathbb{R}^m_>$ and for the normalized weight space $\widetilde{\mathcal{W}}^0$.

3.1.4 Arrangements of hyperplanes

A *hyperplane* h in $\mathbb{R}^m$ is defined as the affine hull of m affinely independent points. Given a finite set of hyperplanes $H = \{h_1, \ldots, h_n\}$, the hyperplanes subdivide $\mathbb{R}^m$ into connected polytopes of different dimensions. This is called the *arrangement of hyperplanes* [see Edelsbrunner, 1987]. Every hyperplane h_i subdivides $\mathbb{R}^m$ into two open half-spaces h_i^+ and h_i^-, where the allocation of half-spaces to identifiers h_i^+ and h_i^- is arbitrary but fixed and assumes to be given in the context of a problem instance. For a point λ in $\mathbb{R}^m$ we define the *position vector of* λ as $\mathrm{Pos}(\lambda) = (\mathrm{Pos}_1(\lambda), \ldots, \mathrm{Pos}_n(\lambda))$ with

$$\mathrm{Pos}_i(\lambda) = \begin{cases} -1 & \text{if } \lambda \in h_i^- \\ 0 & \text{if } \lambda \in h_i \\ +1 & \text{if } \lambda \in h_i^+ \end{cases}$$

for $i = 1, \ldots, n$. Two points are called equivalent if their position vectors are equal. This defines an equivalence relation on $\mathbb{R}^m$, where the equivalence classes are called *faces* φ of the arrangement of hyperplanes. Note that the arrangement of hyperplanes is a partition of $\mathbb{R}^m$ and that each point λ is assigned to exactly one face. The set of points belonging to

one face is connected and the position vector $\mathrm{Pos}(\varphi)$ of face φ is set to $\mathrm{Pos}(\varphi) = \mathrm{Pos}(\lambda)$, for an arbitrary point λ in φ. A face of dimension k is called a *k-face* $\varphi^{(k)}$. Furthermore, a 0-face is called a *vertex*, a 1-face is called an *edge*, an $(m-1)$-face is called a *facet*, and an m-face is called a *cell*.

Let $\varphi_1^{(k)}$ and $\varphi_2^{(k-1)}$ be faces of an arrangement of hyperplanes with dimensions k and $k-1$, respectively, where $1 \leq k \leq m$. If $\varphi_2^{(k-1)}$ is contained in the boundary of $\varphi_1^{(k)}$, then $\varphi_2^{(k-1)}$ is called a *subface of* $\varphi_1^{(k)}$. If $\varphi_2^{(k-1)}$ is a subface of $\varphi_1^{(k)}$, consequently, the position vectors $\mathrm{Pos}(\varphi_1^{(k)})$ and $\mathrm{Pos}(\varphi_2^{(k-1)})$ differ in positions $i \in J \subseteq \{1, \ldots, n\}$, with $|J| \geq 1$, and the corresponding entries in the position vector of $\varphi_2^{(k-1)}$ are 0, i. e., $\mathrm{Pos}_i(\varphi_1^{(k)}) = \mathrm{Pos}_i(\varphi_2^{(k-1)})$ for $i \in \{1, \ldots, n\} \setminus J$ and $\mathrm{Pos}_i(\varphi_2^{(k-1)}) = 0$ and $\mathrm{Pos}_i(\varphi_1^{(k)}) \neq 0$ for $i \in J$. In Figure 3.1 an illustrative example of the above definitions is given. We call a pair of faces $\varphi_0^{(\ell)}$ and $\varphi_{k-\ell}^{(k)}$, with $0 \leq \ell < k \leq m$, *adjacent*, if there exists a set of faces $\{\varphi_1^{(\ell+1)}, \ldots, \varphi_{k-\ell-1}^{(k-1)}\}$ such that $\varphi_s^{(\ell+s)}$ is a subface of $\varphi_{s+1}^{(\ell+s+1)}$, for $0 \leq s < k-\ell-1$. This implies that $\varphi_0^{(\ell)}$ is part of the closure of $\varphi_{k-\ell}^{(k)}$.

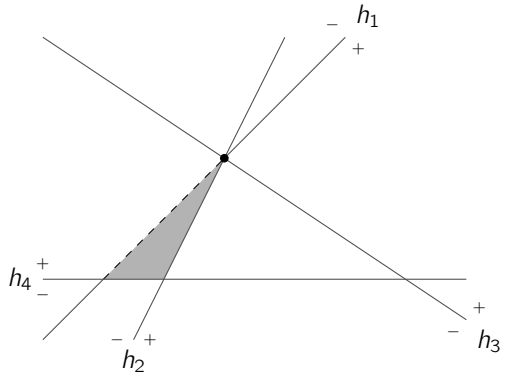


Figure 3.1: Arrangement of four hyperplanes in $\mathbb{R}^2$ with 10 cells (2-faces), 13 facets/edges (1-faces), and 4 vertices (0-faces). The highlighted vertex (•) has the vector $(0, 0, 0, +1)^\top$ as position vector and is a subface of the six surrounding facets. The highlighted facet (dashed line) has the vector $(0, -1, -1, +1)^\top$ as position vector and is subface of the two neighboring cells. The highlighted cell (shaded area) has the vector $(+1, -1, -1, +1)^\top$ as position vector. The highlighted vertex and cell are adjacent.

An arrangement of n hyperplanes in $\mathbb{R}^m$ with $m \leq n$ is called *simple* if the intersection of any subset of m hyperplanes is a unique point and if the intersection of any subset of $(m+1)$ hyperplanes is empty. This implies that the position vectors of $\varphi_1^{(k)}$ and $\varphi_2^{(k-1)}$, $\varphi_2^{(k-1)}$ subface of $\varphi_1^{(k)}$, differ in one single position. The arrangement of Figure 3.1 is not simple, since three hyperplanes, h_1, h_2 and h_3, are intersecting in one point in $\mathbb{R}^2$.

In the following, the number of cells of an arrangement of hyperplanes in $\mathbb{R}^m$ plays an important role. Buck [1943] showed that, for simple arrangements and fixed m, the number of cells is equal to $\sum_{i=0}^{m} \binom{n}{i}$. This number is an upper bound for non-simple arrangements. In general, the number of k-faces is bounded by $\mathcal{O}(n^m)$ for each k, $0 \leq k \leq m$.

An arrangement of n hyperplanes in $\mathbb{R}^m$ with $m \leq n$ is called *central*, if $\mathbf{0}_m$ is contained in every hyperplane. Trivially, unless n is equal to m, central arrangements are not simple and simple arrangements cannot be central. Zaslavsky [1975] showed that, for central arrangements in $\mathbb{R}^m$, m fixed, the number of cells is bounded by

$$2 \cdot \sum_{i=0}^{m-1} \binom{n-1}{i}.$$

Hence, fixing one central intersection point reduces the number of cells to $\mathcal{O}(n^{m-1})$.

Edelsbrunner [1987] presented an algorithm to compute a graph-based representation of the whole structure of an arrangement of hyperplanes, i. e., for representing all faces and all relations between the faces. The algorithm works in $\mathcal{O}(n^m)$ time, which is asymptotically optimal since the number of faces is also in $\mathcal{O}(n^m)$. The space complexity is as large as the output size. Ferrez et al. [2005] presented a reverse search algorithm that identifies all cells of a central arrangement of hyperplanes. The authors take advantage of the centrality of the arrangement to reduce the dimension by one and, hence, work with a general arrangement in $\mathbb{R}^{m-1}$. The algorithm has a time complexity of $\mathcal{O}(nc\ \mathrm{LP}(n, m))$, where c is the number of cells of the arrangement, which is bounded by $\mathcal{O}(n^{m-1})$; $\mathrm{LP}(n, m)$ denotes the complexity for solving a linear program with n inequalities and m variables, which can be done in polynomial time with interior-point methods [see, e. g., Karmarkar, 1984]. This is, in fact, a weaker bound than for the approach by Edelsbrunner [1987]. However, the space complexity of their algorithm is in $\mathcal{O}(nm)$, improving the bound of the approach by Edelsbrunner. Moreover, the authors argue that their algorithm is easier to implement.

3.2 Multi-objective unconstrained combinatorial optimization problems, weight space, zonotopes, and arrangements of hyperplanes

The multi-objective unconstrained combinatorial optimization problem, the weight space, zonotopes and arrangements of hyperplanes are related. For a fixed number m of objective functions, these interrelations reveal a polynomial bound on the number of extreme supported solutions for (mo.0c) with respect to the number of items n:

Theorem 3.3 *The number of extreme supported efficient solutions of* (mo.0c) *with m objective functions and n items is bounded by:*

$$|\mathcal{X}_{eE}| \leq 2 \cdot \sum_{i=0}^{m-1} \binom{n-1}{i},$$

i. e., for fixed m, (mo.0c) *has at most $\mathcal{O}(n^{m-1})$ extreme supported efficient solutions.*

The proof of Theorem 3.3 is given in the following sections. We show that every extreme supported solution of (mo.0c) is related to exactly one cell of an associated central arrangement of hyperplanes. Thus, the result of Zaslavsky [1975] on the bound on the number of cells of a central arrangement of hyperplanes holds also for the number of extreme supported solutions of (mo.0c). We use the concept of zonotopes to show the interrelation of (mo.0c) and arrangements of hyperplanes. The instance of Example 3.4 is introduced for illustration purposes:

Example 3.4

$$\begin{array}{lll} \max & - & x_2 + 3x_3 + 6x_4 - 5x_5 + x_6 \\ \max & x_1 + 2x_2 - 3x_3 - 2x_4 - & x_5 + x_6 \\ \text{s.t.} & x_i \in \{0,1\}, & i = 1, \ldots, 6. \end{array}$$

3.2.1 Multi-objective unconstrained combinatorial optimization problems and zonotopes

Zonotopes and (mo.0c) are related to each other. The LP-relaxation of (mo.0c), where the constraint is relaxed to $x \in [0,1]^n$, is a link between both concepts. Let $\mathcal{Y}^{\text{LP}}$ be the set of feasible points of the LP-relaxation of (mo.0c). We show that the set $\mathcal{Y}^{\text{LP}}$ and the convex hull of the set of feasible points $\text{conv}(\mathcal{Y})$ of (mo.0c) are equal. Let $\{x^1, \ldots, x^{2^n}\} = \mathcal{X} = \{0,1\}^n$ denote the set of all feasible solutions of (mo.0c). Note that $p_i^\bullet \in \mathbb{Z}^m$ is the profit vector of item i, for $i = 1, \ldots, n$, cf. Section 2.2.

$$\begin{aligned} \text{conv}(\mathcal{Y}) &= \left\{ \sum_{k=1}^{2^n} \mu_k f(x^k) \,:\, \sum_{k=1}^{2^n} \mu_k = 1, \mu^k \geq 0, \forall k \in \{1, \ldots, 2^n\} \right\} \\ &= \left\{ \sum_{k=1}^{2^n} \mu_k \sum_{i=1}^{n} x_i^k p_i^\bullet \,:\, \sum_{k=1}^{2^n} \mu_k = 1, \mu^k \geq 0, \forall k \in \{1, \ldots, 2^n\} \right\} \\ &= \left\{ \sum_{i=1}^{n} \left(\sum_{k=1}^{2^n} \mu_k x_i^k \right) p_i^\bullet \,:\, \sum_{k=1}^{2^n} \mu_k = 1, \mu^k \geq 0, \forall k \in \{1, \ldots, 2^n\} \right\} \\ &\overset{(*)}{=} \left\{ \sum_{i=1}^{n} \hat{x}_i p_i^\bullet \,:\, \hat{x}_i \in [0,1], \forall i \in \{1, \ldots, n\} \right\} = \mathcal{Y}^{\text{LP}}. \end{aligned}$$

The equality in $(*)$ holds since (mo.0c) is a binary problem and, therefore, all extreme points of the convex hull of the set of feasible solutions $\text{conv}(\mathcal{X})$ are defined by binary solutions.

For a given instance of (mo.0c), we can define an *associated zonotope*: For each item i of (mo.0c), for $i = 1, \ldots, n$, we define a line segment, using the corresponding profit

vectors $p_i^\bullet$, as $[0, p_i^\bullet] = \{y_i \in \mathbb{R}^m : y_i = \mu p_i^\bullet, \mu \in [0,1]\}$. The zonotope $\mathcal{Z}$ defined by these line segments is equal to the convex hull of the set of feasible points $\text{conv}(\mathcal{Y})$ of (mo.0c):

$$\text{conv}(\mathcal{Y}) = \mathcal{Y}^{\text{LP}} = \left\{ \sum_{i=1}^{n} \hat{x}_i p_i^\bullet : \hat{x}_i \in [0,1], \forall i \in \{1,\dots,n\} \right\}$$

$$= \left\{ \sum_{i=1}^{n} y_i : y_i \in [0, p_i^\bullet], \forall i \in \{1,\dots,n\} \right\} = \mathcal{Z}.$$

In particular, the extreme points of $\mathcal{Z}$ and $\text{conv}(\mathcal{Y})$ are equal.

Example 3.5 *The zonotope in $\mathbb{R}^2$ defined by the line segments*

$$\ell_1 = [(0,0)^\top, (0,1)^\top], \quad \ell_2 = [(0,0)^\top, (-1,2)^\top], \quad \ell_3 = [(0,0)^\top, (3,-3)^\top],$$
$$\ell_4 = [(0,0)^\top, (6,-2)^\top], \quad \ell_5 = [(0,0)^\top, (-5,-1)^\top], \quad \ell_6 = [(0,0)^\top, (1,1)^\top]$$

is equal to the convex hull of the set of feasible points of Example 3.4, see Figure 3.2.

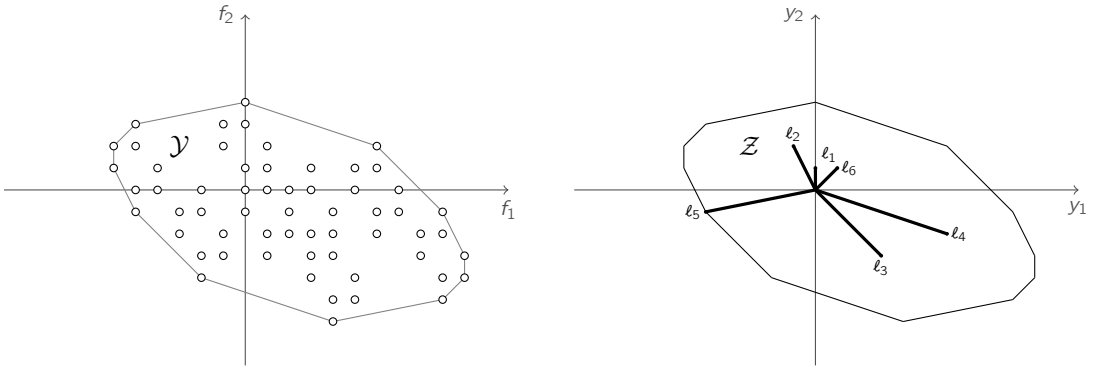


Figure 3.2: On the left: set of feasible points $\mathcal{Y}$ and its convex hull for Example 3.4. On the right: associated zonotope $\mathcal{Z}$ with line segments ℓ_1 to ℓ_6 (cf. Example 3.5).

Conversely, for a given zonotope $\mathcal{Z}$, we can define an associated instance of (mo.0c): For each defining line segment $[u_i, v_i]$, $i \in \{1,\dots,n\}$, we define a profit vector $p_i^\bullet = v_i - u_i$ for item i, for $i = 1,\dots,n$. Additionally, each objective function has a constant term $z^j = \sum_{i=1}^n u_i^j$, for $j = 1,\dots,m$, and we define $z^\bullet = (z^1,\dots,z^m)^\top = \sum_{i=1}^n u_i$. The convex hull of the set of feasible points $\text{conv}(\mathcal{Y})$ of this instance, as well as the corresponding set of feasible points $\mathcal{Y}^{\text{LP}}$ of the LP-relaxation, is equal to the zonotope $\mathcal{Z}$:

$$\mathcal{Z} = \left\{ \sum_{i=1}^{n} y_i : y_i \in [u_i, v_i], \forall i \in \{1,\dots,n\} \right\}$$

$$= \left\{ \sum_{i=1}^{n} y_i : y_i = u_i + \mu_i(v_i - u_i), \mu_i \in [0,1], \forall i \in \{1,\dots,n\} \right\}$$

$$= \left\{ \sum_{i=1}^{n} u_i + \sum_{i=1}^{n} \mu_i (v_i - u_i) \,:\, \mu_i \in [0,1], \forall i \in \{1,\dots,n\} \right\}$$

$$= \left\{ z^\bullet + \sum_{i=1}^{n} \mu_i p_i^\bullet \,:\, \mu_i \in [0,1], \forall i \in \{1,\dots,n\} \right\}$$

$$= \mathcal{Y}^{\mathrm{LP}} = \mathrm{conv}(\mathcal{Y}).$$

In the following, we call an extreme point of a zonotope nondominated if the corresponding point of the associated problem (mo.0c) is nondominated. Note that the corresponding nondominated points of (mo.0c) in the objective space are extreme supported points since they are extreme points of the nondominated frontier.

3.2.2 Arrangements of hyperplanes and zonotopes

It is well known that arrangements of hyperplanes are dual to zonotopes [see e. g., Edelsbrunner, 1987]. Let a zonotope $\mathcal{Z} \subset \mathbb{R}^m$ be defined by n line segments $[0, p_i^\bullet]$ with $p_i^\bullet \in \mathbb{Z}^m$, for $i = 1,\dots,n$. We assume that $m < n$. An *associated arrangement of hyperplanes* can be defined by the hyperplanes

$$h_i = \{\lambda \in \mathbb{R}^m \,:\, \langle p_i^\bullet, \lambda \rangle = 0\}$$

and the corresponding half-spaces

$$h_i^+ = \{\lambda \in \mathbb{R}^m \,:\, \langle p_i^\bullet, \lambda \rangle > 0\}$$
$$h_i^- = \{\lambda \in \mathbb{R}^m \,:\, \langle p_i^\bullet, \lambda \rangle < 0\}$$

for $i = 1,\dots,n$. This arrangement is central, since $\mathbf{0}_n \in h_i$ for all $i = 1,\dots,n$.

Example 3.6 *The zonotope of Example 3.5 is dual to the arrangement of hyperplanes*

$$h_1 = \{\lambda \in \mathbb{R}^2 \,:\, \lambda_2 = 0\}, \qquad h_2 = \{\lambda \in \mathbb{R}^2 \,:\, -\lambda_1 + 2\lambda_2 = 0\},$$
$$h_3 = \{\lambda \in \mathbb{R}^2 \,:\, 3\lambda_1 - 3\lambda_2 = 0\}, \quad h_4 = \{\lambda \in \mathbb{R}^2 \,:\, 6\lambda_1 - 2\lambda_2 = 0\},$$
$$h_5 = \{\lambda \in \mathbb{R}^2 \,:\, -5\lambda_1 - \lambda_2 = 0\}, \quad h_6 = \{\lambda \in \mathbb{R}^2 \,:\, \lambda_1 + \lambda_2 = 0\},$$

which is central, see Figure 3.3.

3.2.3 Multi-objective unconstrained combinatorial optimization problems, weight space and arrangements of hyperplanes

As mentioned above, extreme supported points of (mo.0c) can be computed using the weighted sum scalarization, where the weights are in $\mathbb{R}^m_>$. The objective function of the weighted sum problem (WS(λ)) can be reorganized as follows:

$$\sum_{j=1}^{m} \lambda_j \cdot f_j(x) = \sum_{j=1}^{m} \lambda_j \cdot \left(\sum_{i=1}^{n} p_i^j x_i \right) = \sum_{i=1}^{n} \left(\sum_{j=1}^{m} \lambda_j p_i^j \right) x_i = \sum_{i=1}^{n} \langle p_i^\bullet, \lambda \rangle x_i.$$

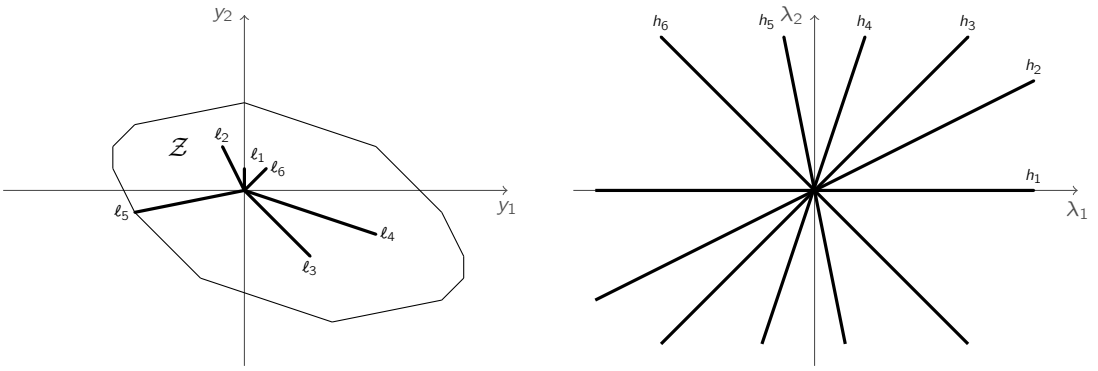


Figure 3.3: On the left: zonotope $\mathcal{Z}$ with line segments ℓ_1 to ℓ_6 (cf. Example 3.5). On the right: associated dual arrangement of hyperplanes with hyperplanes h_1 to h_6 (cf. Example 3.6).

The coefficients $\langle p_i^\bullet, \lambda \rangle$, for $i = 1, \ldots, n$, define an arrangement of hyperplanes in $\mathbb{R}^m_>$.

Recall that the optimal solution of (1o.0c) can be built by deciding on each variable separately depending on the sign of the coefficient. Thus, for item i, for $i = 1, \ldots, n$, the position of the weight λ in $\mathbb{R}^m_>$ defines the optimal choice for x_i: If λ is in h_i^+, $x_i = 1$ is optimal, if λ is in h_i^-, $x_i = 0$ is optimal, and if λ is in h_i, both alternatives are optimal. This last case indicates that, as we know from the weighted sum scalarization, one weight λ can correspond to several nondominated points that define a face of the nondominated frontier. All weights $\lambda \in \mathbb{R}^m_>$ with equal position vectors correspond to the same set of nondominated points.

This shows that the duality of (mo.0c) (zonotopes) and the arrangement of hyperplanes has an order reversing characteristic: A nondominated k-face of the convex hull of feasible points in $\mathbb{R}^m$ corresponds to an $(m-k)$-face of the associated arrangement of hyperplanes, in $\mathbb{R}^m_>$. Thus, extreme supported points of (mo.0c) correspond to cells of the associated arrangement and vice versa. Since the number of cells in the arrangement is bounded, the same bound holds for the number of extreme supported points of (mo.0c). Furthermore, we know that a cell of the arrangement is either in h_i^+ or in h_i^-, for all $i = 1, \ldots, n$. Thus, either $x_i = 0$ or $x_i = 1$ is optimal in the corresponding solution, but not both alternatives. Thus, the cell corresponds to one unique solution of (mo.0c).

Corollary 3.7 *Every extreme supported nondominated point of* (mo.0c) *is realized by exactly one extreme supported efficient solution.*

The arrangement of hyperplanes h_i, for all $i = 1, \ldots, n$, can be used to define the decomposition of the weight space $\mathbb{R}^m_>$ for (mo.0c). Given an extreme supported solution x and the corresponding point $y = f(x)$, the set $\mathcal{W}^0(y)$ consists of all faces of the arrangement of hyperplanes that correspond to the solution x. These faces are the cell of the arrangement corresponding to x and all adjacent faces in its boundary. We use the term "weight space decomposition" also for the arrangement of hyperplanes in the remainder of

this chapter since the arrangement is in fact a decomposition of the weight space and the weight space decomposition defined as in Section 3.1.3 can be determined knowing the arrangement of hyperplanes.

The centrality of the arrangement confirms that the reduction from the weight space $\mathbb{R}^m_>$ to the normalized weight space $\widetilde{\mathcal{W}}^0$ is justified for the weighted sum scalarization (cf. Section 2.1): Every cell of the arrangement in $\mathbb{R}^m_>$ intersects $\widetilde{\mathcal{W}}^0$ and, hence, every extreme supported point is represented in this intersection. The decomposition of the normalized weight space $\widetilde{\mathcal{W}}^0$ is still defined by an arrangement of hyperplanes. Due to the condition that the sum of weights λ_i, for $i = 1, \ldots, n$, should be equal to 1 this arrangement is not central.

The original arrangement of hyperplanes can also be projected on $\mathbb{R}^{m-1}$, such that the projected weight space $\mathcal{W}^0$ is subdivided by this arrangement of hyperplanes. We call the arrangement in the projected weight space the *associated projected arrangement* of (mo.0c). A further discussion follows in Section 3.3.

Example 3.8 *The instance of Example 3.4 has four extreme supported points. Hence, the associated arrangement of hyperplanes has four corresponding cells intersecting with the first quadrant of* $\mathbb{R}^2$*. Also the normalized weight space* $\widetilde{\mathcal{W}}^0$ *and the projected weight space* $\mathcal{W}^0$ *are subdivided into four segments by the associated projected arrangement, cf. Figure 3.4.*

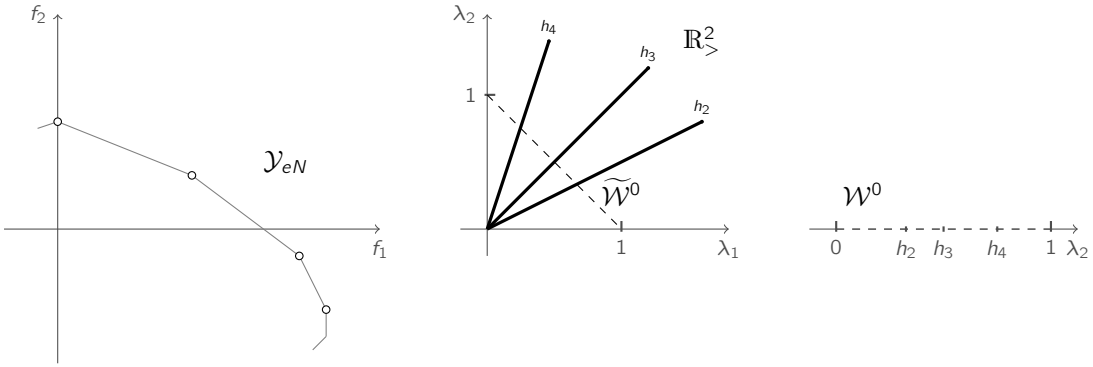


Figure 3.4: On the left: extreme supported points of Example 3.4. In the middle: intersection of the associated arrangement of hyperplanes with the first quadrant in $\mathbb{R}^2$ and intersection with the normalized weight space $\widetilde{\mathcal{W}}^0$. On the right: associated projected arrangement of hyperplanes in the projected weight space $\mathcal{W}^0$.

Consider again the complete arrangement of hyperplanes. Each cell of the arrangement corresponds to an extreme point of the associated zonotope and vice versa. Each orthant of $\mathbb{R}^m$ corresponds to a combination of maximization and minimization objectives and the associated notion of dominance. To be more precise: If the weight value λ_j, for $j \in \{1, \ldots, m\}$, is positive, then the corresponding extreme points of the zonotope are nondominated for maximizing objective function $f_j(x)$. If the weight value λ_j, for

$j \in \{1, \ldots, m\}$, is negative, then the corresponding extreme points of the zonotope are nondominated for minimizing objective function $f_j(x)$.

Example 3.9 *Consider the following modification of Example 3.4 where we switched* max *to* min *in the second objective:*

$$\begin{array}{lll} \max & - & x_2 + 3x_3 + 6x_4 - 5x_5 + x_6 \\ \min & x_1 + 2x_2 - 3x_3 - 2x_4 - & x_5 + x_6 \\ \text{s.t.} & x_i \in \{0,1\}, & i = 1, \ldots, 6. \end{array}$$

In Figure 3.5, the convex hull of feasible points for this instance of (mo.0c) *is shown. The symbols • highlight three extreme points that are nondominated w. r. t. the maximization of the first and minimization of the second objective. The corresponding part of the arrangement of hyperplanes is inside the second quadrant of* $\mathbb{R}^2$*, i. e., cells intersecting with* $\{(\lambda_1, \lambda_2) \in \mathbb{R}^2 : \lambda_1 > 0, \lambda_2 < 0\}$.

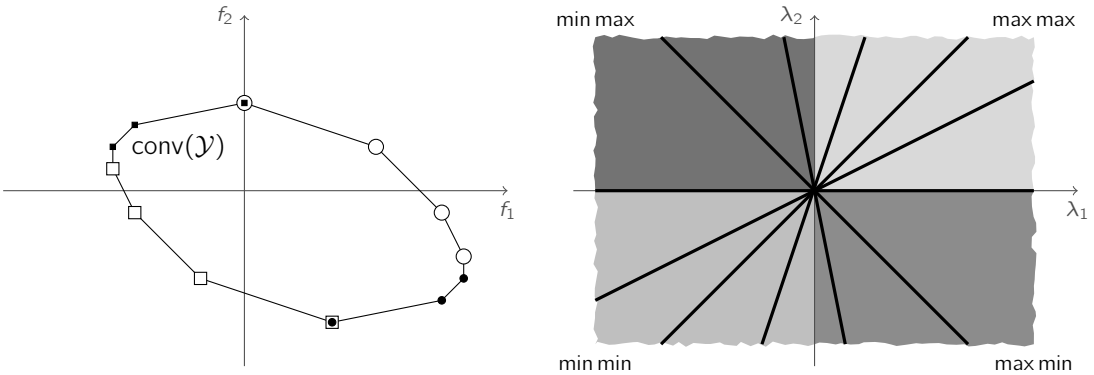


Figure 3.5: On the left: extreme points of the convex hull of the feasible set conv($\mathcal{Y}$) of Example 3.4, where the symbols ○ indicate extreme supported points and the symbols □, •, and ▪ indicate extreme supported points if the objectives would be changed to $(\min f_1, \min f_2)$, $(\max f_1, \min f_2)$, and $(\min f_1, \max f_2)$, respectively. On the right: associated arrangement of hyperplanes in $\mathbb{R}^2$ corresponding to the associated zonotope and parts of the arrangement of hyperplanes that have to be considered for the respective notion of nondominance.

In this context, it is very intuitive to see that switching from maximization to minimization in all objective functions generates the same number of nondominated points where the efficient solutions with respect to maximization are reverse to the efficient solutions with respect to minimization. Since the arrangement of hyperplanes is central, all cells reappear in the opposite orthant of $\mathbb{R}^m$ with reversed position vector. Thus, the associated zonotope and the convex hull of the feasible set of the corresponding instance of (mo.0c) are centrally symmetric.

3.2.4 Summary

The following table summarizes the interrelations between (mo.0c), zonotopes and the associated arrangements of hyperplanes:

cells of central arrangement of hyperplanes in $\mathbb{R}^m$	$\leftrightarrow$	extreme points of zonotope in $\mathbb{R}^m$	$\leftrightarrow$	extreme points of conv($\mathcal{Y}$) in $\mathbb{R}^m$
$\cup$		$\cup$		$\cup$
cells of central arrangement of hyperplanes in $\mathbb{R}^m_>$	$\leftrightarrow$	nondominated extreme points of zonotope in $\mathbb{R}^m$	$\leftrightarrow$	extreme supported points $\mathcal{Y}_{eN}$ in $\mathbb{R}^m$
$\cup$	$\nearrow\!\!\swarrow$			
cells of projected arrangement of hyperplanes in $\mathcal{W}^0 \subset \mathbb{R}^{m-1}_>$				

We can conclude that the number of extreme supported efficient solutions of (mo.0c) is equal to the number of extreme supported nondominated points of (mo.0c). This number is again equal to the number of nondominated extreme points of the associated zonotope and also equal to the number of cells of the arrangement of hyperplanes associated to (mo.0c). In particular, the bound on the number of cells in the arrangement of hyperplanes also holds for the extreme supported solutions of (mo.0c), which proves Theorem 3.3.

In Section 3.3 we explain these interrelations with regard to (mo.0c) in more detail and give ideas, remarks and an algorithm for an efficient computation of extreme supported solutions. We emphasize that the statements of this section are no longer true when a constraint is added to the problem formulation. Nevertheless, we demonstrate possibilities to use these results for (mo.1c) in Section 3.4.

3.3 Efficient computation of extreme supported solutions of multi-objective unconstrained combinatorial optimization problems

In Section 3.2, the interrelations between (mo.0c), zonotopes, arrangements of hyperplanes, and the weight space were presented. In this section, we review these connections from a more problem specific perspective and, based on that, we introduce an algorithm to efficiently compute the extreme supported solutions.

At first, the concepts are only applied to the bi-objective case. The ideas can be explained and visualized in two dimensions very well and, afterward, the generalization to more objectives can be done straightforward, which is done in Section 3.3.2. We present an algorithm for the special case of $m = 3$ where all coefficients of the first objective function

are positive and all coefficients of the second and third objective function are negative in Section 3.3.3. In Section 3.3.4, the algorithm is generalized to an arbitrary number of objective functions and arbitrary combinations of positive and negative weights. Finally, Section 3.3.5 describes a further generalization of the algorithm for the computation of the extreme points of a zonotope in $\mathbb{R}^m$.

3.3.1 Bi-objective unconstrained combinatorial optimization problems

In order to give a specific insight into the concepts explained in the previous section, we start with the bi-objective unconstrained combinatorial optimization problem (2o.0c) and have a closer look at the weighted sum scalarization of this problem.

As described above, the weighted sum scalarization can be used to compute all extreme supported solutions of (2o.0c). In the bi-objective case, choosing weights $\lambda \in \widetilde{\mathcal{W}}^0 = \left\{\lambda \in \mathbb{R}^2_> \,:\, \lambda_1 + \lambda_2 = 1\right\}$ reduces to choose values λ_2 in the projected weight space $\mathcal{W}^0 = (0,1)$, and setting $\lambda_1 = 1 - \lambda_2$.

As explained in Theorem 3.2, the decision on each item can be made separately for (1o.0c). Based on that, the weight space decomposition can be built. Therefore, we introduce a function $\rho(\lambda_2)$ that measures the utility of item i depending on the weight λ_2:

Definition 3.10 *Let $i \in \{1,\dots,n\}$. The* weighted utility of item i *is defined as $\rho_i : \mathbb{R} \to \mathbb{R}$ with*

$$\rho_i(\lambda_2) = (1-\lambda_2)\cdot p_i^1 + \lambda_2 \cdot p_i^2 = (p_i^2 - p_i^1)\cdot \lambda_2 + p_i^1.$$

If $p_i^1 \neq p_i^2$, the root of $\rho_i(\lambda_2)$, denoted by $\lambda^0_{2,i}$, is called the root weight *of item i, i. e.,*

$$\lambda^0_{2,i} = \frac{-p_i^1}{p_i^2 - p_i^1}.$$

Note that $\rho_i(\lambda_2)$ is a linear function of λ_2 that changes its sign at the root weight $\lambda^0_{2,i}$. Lemma 3.11 describes the dependency of $\lambda^0_{2,i}$ and of the slope of $\rho_i(\lambda_2)$ on p_i^1 and p_i^2.

Lemma 3.11 *Let $i \in \{1,\dots,n\}$. The weighted utility ρ_i depends on the coefficients p_i^1 and p_i^2 in the following way:*

(1) *$\rho_i(\lambda_2)$ has a positive slope iff $0 < p_i^2 - p_i^1$.*

(2) *$\rho_i(\lambda_2)$ is a constant function iff $p_i^1 = p_i^2$.*

(3) *$\rho_i(\lambda_2)$ has a negative slope iff $p_i^2 - p_i^1 < 0$.*

Now let $p_i^1 \neq p_i^2$. Then

(I) *$1 \leq \lambda^0_{2,i}$ iff $\operatorname{sgn}(p_i^1) = \operatorname{sgn}(p_i^2)$ and $0 \leq |p_i^2| < |p_i^1|$.*

(II) *$0 < \lambda^0_{2,i} < 1$ iff $\operatorname{sgn}(p_i^1) \neq \operatorname{sgn}(p_i^2)$ and $p_i^1 \neq 0$ and $p_i^2 \neq 0$.*

(III) *$\lambda^0_{2,i} \leq 0$ iff $\operatorname{sgn}(p_i^1) = \operatorname{sgn}(p_i^2)$ and $0 \leq |p_i^1| < |p_i^2|$.*

Proof.

(1)-(3) Since the weighted utility ρ_i can be rewritten as $\rho_i(\lambda_2) = (p_i^2 - p_i^1) \cdot \lambda_2 + p_i^1$, statements 1.-3. follow immediately.

(I)
$$
\begin{aligned}
& 1 \leq \lambda_{2,i}^0 = \frac{-p_i^1}{p_i^2 - p_i^1} \\
\Leftrightarrow\ & \left[(0 < -p_i^1) \text{ and } (0 < p_i^2 - p_i^1) \text{ and } (p_i^2 - p_i^1 \leq -p_i^1)\right] \\
& \quad \text{or } \left[(-p_i^1 < 0) \text{ and } (p_i^2 - p_i^1 < 0) \text{ and } (-p_i^1 \leq p_i^2 - p_i^1)\right] \\
\Leftrightarrow\ & \left[(p_i^1 < 0) \text{ and } (p_i^1 < p_i^2) \text{ and } (p_i^2 \leq 0)\right] \text{ or } \left[(0 < p_i^1) \text{ and } (p_i^2 < p_i^1) \text{ and } (0 \leq p_i^2)\right] \\
\Leftrightarrow\ & \left[(p_i^1 < p_i^2 \leq 0)\right] \text{ or } \left[(0 \leq p_i^2 < p_i^1)\right] \\
\Leftrightarrow\ & (\operatorname{sgn}(p_i^1) = \operatorname{sgn}(p_i^2)) \text{ and } (0 \leq |p_i^2| < |p_i^1|).
\end{aligned}
$$

(II)
$$
\begin{aligned}
& 0 < \lambda_{2,i}^0 = \frac{-p_i^1}{p_i^2 - p_i^1} < 1 \\
\Leftrightarrow\ & \left[(0 < -p_i^1) \text{ and } (0 < p_i^2 - p_i^1) \text{ and } (-p_i^1 < p_i^2 - p_i^1)\right] \\
& \quad \text{or } \left[(-p_i^1 < 0) \text{ and } (p_i^2 - p_i^1 < 0) \text{ and } (p_i^2 - p_i^1 < -p_i^1)\right] \\
\Leftrightarrow\ & \left[(p_i^1 < 0) \text{ and } (p_i^1 < p_i^2) \text{ and } (0 < p_i^2)\right] \text{ or } \left[(0 < p_i^1) \text{ and } (p_i^2 < p_i^1) \text{ and } (p_i^2 < 0)\right] \\
\Leftrightarrow\ & \left[(p_i^1 < 0 < p_i^2)\right] \text{ or } \left[(p_i^2 < 0 < p_i^1)\right] \\
\Leftrightarrow\ & (\operatorname{sgn}(p_i^1) \neq \operatorname{sgn}(p_i^2)) \text{ and } (p_i^1 \neq 0) \text{ and } (p_i^2 \neq 0).
\end{aligned}
$$

(III)
$$
\begin{aligned}
& \lambda_{2,i}^0 = \frac{-p_i^1}{p_i^2 - p_i^1} \leq 0 \\
\Leftrightarrow\ & \left[(-p_i^1 \leq 0) \text{ and } (0 < p_i^2 - p_i^1)\right] \text{ or } \left[(0 \leq -p_i^1) \text{ and } (p_i^2 - p_i^1 < 0)\right] \\
\Leftrightarrow\ & \left[(0 \leq p_i^1) \text{ and } (p_i^1 < p_i^2)\right] \text{ or } \left[(p_i^1 \leq 0) \text{ and } (p_i^2 < p_i^1)\right] \\
\Leftrightarrow\ & \left[(0 \leq p_i^1 < p_i^2)\right] \text{ or } \left[(p_i^2 < p_i^1 \leq 0)\right] \\
\Leftrightarrow\ & (\operatorname{sgn}(p_i^1) = \operatorname{sgn}(p_i^2)) \text{ and } (0 \leq |p_i^1| < |p_i^2|).
\end{aligned}
$$

□

The results of Lemma 3.11 induce an explicit scheme for choosing the variables x_i to obtain optimal solutions for given weights λ_2 in $\mathcal{W}^0$. Table 3.1 presents all possible cases with respect to the coefficients p_i^1 and p_i^2 and the respective consequences for the sign of the weighted utility ρ_i.

Corollary 3.12 *Let $\bar{\lambda}_2 \in \mathcal{W}^0$. For (2o.0c), all optimal solutions of the weighted sum scalarization for $\bar{\lambda}_2$ can be determined using the scheme of Table 3.1 by setting*

$$
x_i \begin{cases} = 0 & \text{if } \rho_i(\bar{\lambda}_2) < 0 \\ \in \{0,1\} & \text{if } \rho_i(\bar{\lambda}_2) = 0 \\ = 1 & \text{if } \rho_i(\bar{\lambda}_2) > 0 \end{cases}
$$

for $i = 1, \ldots, n$.

coefficients	case		sign of $\rho_i(\lambda)$
$0 < p_i^1 = p_i^2$	(2)	$\Rightarrow$	$\rho_i(\lambda_2) > 0$ for all $\lambda_2 \in \mathcal{W}^0$
$p_i^1 = p_i^2 < 0$	(2)	$\Rightarrow$	$\rho_i(\lambda_2) < 0$ for all $\lambda_2 \in \mathcal{W}^0$
$0 \leq p_i^2 < p_i^1$	(3) and (I)	$\Rightarrow$	$\rho_i(\lambda_2) > 0$ for all $\lambda_2 \in \mathcal{W}^0$
$p_i^1 < p_i^2 \leq 0$	(1) and (I)	$\Rightarrow$	$\rho_i(\lambda_2) < 0$ for all $\lambda_2 \in \mathcal{W}^0$
$p_i^2 < 0 < p_i^1$	(3) and (II)	$\Rightarrow$	$\begin{cases} \rho_i(\lambda_2) > 0 & \text{for all } \lambda_2 \in (0, \lambda_{2,i}^0) \\ \rho_i(\lambda_2) = 0 & \text{for } \lambda_2 = \lambda_{2,i}^0 \\ \rho_i(\lambda_2) < 0 & \text{for all } \lambda_2 \in (\lambda_{2,i}^0, 1) \end{cases}$
$p_i^1 < 0 < p_i^2$	(1) and (II)	$\Rightarrow$	$\begin{cases} \rho_i(\lambda_2) < 0 & \text{for all } \lambda_2 \in (0, \lambda_{2,i}^0) \\ \rho_i(\lambda_2) = 0 & \text{for } \lambda_2 = \lambda_{2,i}^0 \\ \rho_i(\lambda_2) > 0 & \text{for all } \lambda_2 \in (\lambda_{2,i}^0, 1) \end{cases}$
$0 \leq p_i^1 < p_i^2$	(1) and (III)	$\Rightarrow$	$\rho_i(\lambda_2) > 0$ for all $\lambda_2 \in \mathcal{W}^0$
$p_i^2 < p_i^1 \leq 0$	(3) and (III)	$\Rightarrow$	$\rho_i(\lambda_2) < 0$ for all $\lambda_2 \in \mathcal{W}^0$

Table 3.1: Possible combinations of coefficients p_i^1 and p_i^2 and implications for the weighted utility ρ_i.

Proof. The weighted sum scalarization of (2o.0c) for $\bar{\lambda}_2$ is a single-objective unconstrained combinatorial optimization problem, cf. Theorem 3.2. □

Summarizing the discussion above, all root weights $\lambda_{2,i}^0$ with $0 < \lambda_{2,i}^0 < 1$, for $i = 1, \ldots, n$, divide the projected weight space into intervals $I_t \subset \mathcal{W}^0$, see Example 3.14 and Figure 3.6. Each interval corresponds to one specific combination of signs of the weighted utilities $\rho_i(\lambda_2)$ and, therefore, to one extreme supported solution of (2o.0c). If a specific weight λ_2 is equal to a root weight, both solutions of the adjacent intervals in the projected weight space are optimal for the corresponding weighted sum objective. If two or more root weights are equal, the corresponding weighted sum objective has more than two optimal solutions.

Remark 3.13 *Nonextreme supported solutions occur if two or more items have weighted utility values $\rho_i(\bar{\lambda}_2) = 0$ for on specific weight value $\bar{\lambda}_2 \in \mathcal{W}^0$. This results if the root weights are equal, which occurs if the coefficients for items i and j are linearly dependent, i. e., if there exists $\alpha \in \mathbb{R}\backslash\{0\}$ such that $p_i^1 = \alpha \cdot p_j^1$ and $p_i^2 = \alpha \cdot p_j^2$. All possible combinations of including or not including those items lead to equal objective function values in the weighted sum objective $f_{\bar{\lambda}}(x)$. The two selections corresponding to the adjacent intervals are extreme supported solutions, all other possible selections are nonextreme supported solutions.*

Example 3.14 *We solve the following bi-objective unconstrained combinatorial optimization problem:*

$$\begin{array}{lll} \max & - & x_2 + 3x_3 - 4x_4 + 6x_5 - 5x_6 + x_7 \\ \max & & x_1 + 2x_2 - 3x_3 + 4x_4 - 2x_5 - x_6 + x_7 \\ \text{s.t.} & & x_i \in \{0,1\}, \qquad i = 1,\dots,7. \end{array}$$

First, we define the weighted utilities $\rho_i(\lambda_2)$, for all items $i = 1,\dots,7$, and compute the root weights $\lambda^0_{2,i}$:

i	1	2	3	4	5	6	7
$\rho_i(\lambda_2)$	λ_2	$3\lambda_2 - 1$	$-6\lambda_2 + 3$	$8\lambda_2 - 4$	$-8\lambda_2 + 6$	$4\lambda_2 - 5$	$0\lambda_2 + 1$
$\lambda^0_{2,i}$	0	$1/3$	$1/2$	$1/2$	$3/4$	$5/4$	–
case	*(1), (III)*	*(1), (II)*	*(3), (II)*	*(1), (II)*	*(3), (II)*	*(1), (I)*	*(2)*

The root weights divide the projected weight space into four intervals $I_1 = (0, \frac{1}{3})$, $I_2 = (\frac{1}{3}, \frac{1}{2})$, $I_3 = (\frac{1}{2}, \frac{3}{4})$, and $I_4 = (\frac{3}{4}, 1)$, see Figure 3.6.

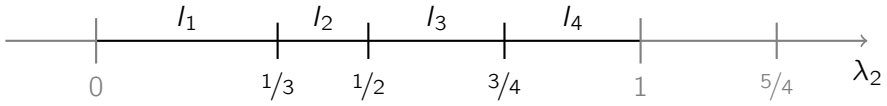


Figure 3.6: Projected weight space $\mathcal{W}^0$ for Example 3.14. The root weights $\lambda^0_{2,i}$ define four segmenting intervals.

For each item and interval, the optimal value of the variable for the weighted problem can be chosen following the scheme of Table 3.1. Variables corresponding to items with coefficients p^1_i and p^2_i of equal sign in both objectives, in this example items 1, 6 and 7, can be fixed apriori in a preprocessing step. The corresponding root weights $\lambda^0_{2,i}$ are not inside the projected weight space $\mathcal{W}^0$ and, consequently, the optimal choice for these variables is equal for all supported solutions. Thus, only variables corresponding to items with $\operatorname{sgn}(p^1_i) \neq \operatorname{sgn}(p^2_i)$ are critical for a solution approach.

All supported solutions are listed in Table 3.2 as well as the corresponding objective function values. Figure 3.7 illustrates the supported points in the objective space. Items 3 and 4 have the same root weight. Hence, for weight $\lambda_2 = 1/2$ four different solutions have equal weighted sum objective function values $f_\lambda(x)$. All four solutions are supported for the bi-objective unconstrained combinatorial optimization problem, but two of them are nonextreme (see Figure 3.7). The root weights $\lambda^0_{2,2}$ and $\lambda^0_{2,5}$ are not equal to any other root weight and, therefore, correspond to two extreme supported solutions.

Example 3.14 shows that it is sufficient to investigate the root weights $\lambda^0_{2,i}$ within the projected weight space to find all supported solutions of (2o.0c). On the one hand, items

		i	1	2	3	4	5	6	7	$f_1(x)$	$f_2(x)$
I_1:	$\lambda_2 \in (0, 1/3)$	$x_i^1 =$	1	0	1	0	1	0	1	10	-3
$\lambda^0_{2,2}$:	$\lambda_2 = 1/3$	$x_i^1 =$	1	0	1	0	1	0	1	10	-3
		$x_i^2 =$	1	1	1	0	1	0	1	9	-1
I_2:	$\lambda_2 \in (1/3, 1/2)$	$x_i^2 =$	1	1	1	0	1	0	1	9	-1
$\lambda^0_{2,3}$:	$\lambda_2 = 1/2$	$x_i^2 =$	1	1	1	0	1	0	1	9	-1
$(= \lambda^0_{2,4})$		$x_i^3 =$	1	1	0	0	1	0	1	6	2
		$x_i^4 =$	1	1	1	1	1	0	1	5	3
		$x_i^5 =$	1	1	0	1	1	0	1	2	6
I_3:	$\lambda_2 \in (1/2, 3/4)$	$x_i^5 =$	1	1	0	1	1	0	1	2	6
$\lambda^0_{2,5}$:	$\lambda_2 = 3/4$	$x_i^5 =$	1	1	0	1	1	0	1	2	6
		$x_i^6 =$	1	1	0	1	0	0	1	-4	8
I_4:	$\lambda_2 \in (3/4, 1)$	$x_i^6 =$	1	1	0	1	0	0	1	-4	8

Table 3.2: Supported solutions and supported points for Example 3.14 corresponding to weights λ_2.

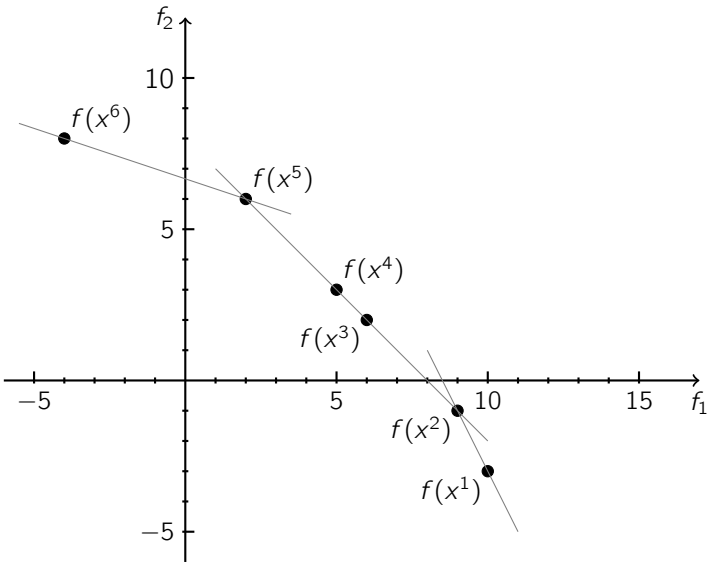


Figure 3.7: Objective space with supported points for Example 3.14. Clearly, points $f(x^3)$ and $f(x^4)$ are supported but nonextreme.

with root weights not in $\mathcal{W}^0$ are included or not included for all weights λ_2 in $\mathcal{W}^0$, and on the other hand, weights λ_2 in intervals $I_t \subset \mathcal{W}^0$ between two consecutive root weights lead to solutions that are also optimal for weighted sum problems for these two root weights.

Algorithm 3.1 gives a possible procedure to compute all extreme supported solutions of (2o.0c). In the first for-loop, an initial extreme supported solution is generated (Lines 3 to 6), starting at the left-hand boundary of the projected weight space with a sufficiently small value $\bar{\lambda}_2 > 0$, i. e., smaller than the smallest positive root weight: $\bar{\lambda}_2 < \min\{\lambda^0_{2,i} : \lambda^0_{2,i} > 0, i = 1, \ldots, n\}$. Consequently, if p^1_i is positive, then the weighted utility $\rho_i(\bar{\lambda}_2) = (1 - \bar{\lambda}_2) \cdot p^1_i + \bar{\lambda}_2 \cdot p^2_i$ is also positive for $\bar{\lambda}_2$ and the variable x_i is set to 1. If p^1_i is equal to zero, then the coefficient of the second objective function decides on the sign of the weighted utility $\rho_i(\lambda_2) = \lambda_2 \cdot p^2_i$. Thus, if p^2_i is positive, then $\rho_i(\lambda_2)$ is also positive for all values of $\lambda_2 \in \mathcal{W}^0$ and we set $x_i = 1$. Every other case leads to a negative weighted utility for $\bar{\lambda}_2$ and we set $x_i = 0$.

All root weights are computed in the first for-loop (Lines 7 to 10). If the root weight of item i, $i = 1, \ldots, n$, is not defined, i. e., if $p^1_i = p^2_i$, the algorithm sets $\lambda^0_{2,i} = 2$. By doing so, the value of x_i will not be further changed.

Let $\mathcal{S}_n$ denote the symmetric group of order n. The root weights are sorted in non-increasing order and $\pi \in \mathcal{S}_n$ denotes the corresponding permutation of the numbers $1, \ldots, n$. Starting from the initial solution, new extreme supported solutions are generated by tracing the root weights with $\lambda^0_{2,\pi(i)} > 0$. At each iteration, i. e., at each root weight $\lambda^0_{2,\pi(i)}$ the value of the current variable $x_{\pi(i)}$ is switched to $1 - x_{\pi(i)}$. If the subsequent root weight $\lambda^0_{2,\pi(i+1)}$ differs from the current root weight $\lambda^0_{2,\pi(i)}$, the newly generated solution x is the extreme supported solution corresponding to the interval right-hand of $\lambda^0_{2,\pi(i)}$ in the projected weight space and can be stored. Otherwise, i. e., if $\lambda^0_{2,\pi(i)} = \lambda^0_{2,\pi(i+1)}$, the current solution is nonextreme supported and, thus, omitted. The algorithm continues with the next index. The while-loop stops as soon as the first root weight $\lambda^0_{2,\pi(i)}$ is greater than or equal to 1.

The algorithm has a worst case complexity of $\mathcal{O}(n \log n)$, since the set of root weights has to be sorted once. The while-loop requires $\mathcal{O}(n)$ because every variable x_i is changed at most once. If the algorithm should be adapted to compute all supported solutions, i. e., including nonextreme supported solutions, this can easily be done. In case of equal root weights, all possible combinations of including or not including the respective items have to be generated. If t items have the same root weight, there exist 2^t corresponding supported solutions. Hence, the adapted algorithm has a worst case complexity of $\mathcal{O}(2^n)$. However, the next supported solution can always be found in constant time. The complexity of the algorithm is, therefore, determined by the number of supported solutions.

Arrangement of hyperplanes In the following, we draw the connection to the results of Section 3.2. The decomposition of the projected weight space $\mathcal{W}^0$ can also be interpreted as the associated projected arrangement of hyperplanes to (2o.0c). Every root weight $\lambda^0_{2,i}$ defines a hyperplane h_i in $\mathbb{R}$ and the half-spaces h^-_i and h^+_i are defined by the sign of

Algorithm 3.1 Algorithm for generating all extreme supported solutions of (2o.0c).

Input: coefficients p_i^1, p_i^2, for $i = 1, \ldots, n$.

$\mathcal{X}_{eE} := \emptyset$, $x := \mathbf{0}_n$
for $i := 1, \ldots, n$ **do** // *compute initial solution and root weights*
  if $p_i^1 > 0$ **then**
    $x_i := 1$
  else if $p_i^1 = 0$ **and** $p_i^2 > 0$ **then**
    $x_i := 1$
  if $p_i^1 = p_i^2$ **then** // *for $p_i^1 = p_i^2$: x_i fixed for all $\lambda_2 \in \mathcal{W}^0$*
    $\lambda^0_{2,i} := 2$
  else
    $\lambda^0_{2,i} := \frac{-p_i^1}{p_i^2 - p_i^1}$
$\mathcal{X}_{eE} := \mathcal{X}_{eE} \cup \{x\}$
compute permutation $\pi \in \mathcal{S}_n$ such that $\lambda^0_{2,\pi(1)} \leq \ldots \leq \lambda^0_{2,\pi(n)}$
$i := 1$, stop $:= 0$
while stop $= 0$ **do** // *compute further solutions*
  if $\lambda^0_{2,\pi(i)} \geq 1$ **then**
    stop $:= 1$
  else if $\lambda^0_{2,\pi(i)} > 0$ **then**
    $x_{\pi(i)} := 1 - x_{\pi(i)}$
    if $\lambda^0_{2,\pi(i)} \neq \lambda^0_{2,\pi(i+1)}$ **then**
      $\mathcal{X}_{eE} := \mathcal{X}_{eE} \cup \{x\}$
  $i := i + 1$

Output: $\mathcal{X}_{eE}$

the corresponding weighted utility, i. e., the entries of the position vector of λ_2 in $\mathbb{R}$ are defined by

$$\mathrm{Pos}_i(\lambda_2) = \begin{cases} -1 & \text{if } \lambda_2 \in h_i^- \quad \Leftrightarrow \ \rho_i(\lambda_2) < 0 \\ 0 & \text{if } \lambda_2 = \lambda_{2,i}^0 \in h_i \quad \Leftrightarrow \ \rho_i(\lambda_2) = 0 \\ +1 & \text{if } \lambda_2 \in h_i^+ \quad \Leftrightarrow \ \rho_i(\lambda_2) > 0 \end{cases}$$

for $i = 1, \ldots, n$. If $\lambda_{2,i}^0$ is not defined, i. e., if $p_i^1 = p_i^2$, then $h_i = \emptyset$ and either $h_i^- = \mathbb{R}$ and $h_i^+ = \emptyset$ or $h_i^- = \emptyset$ and $h_i^+ = \mathbb{R}$, depending on the sign of p_i^1.

Another interesting perspective is offered by the associated arrangement of hyperplanes to (2o.0c) in $\mathbb{R}^2$. To distinguish between the hyperplanes in $\mathbb{R}^2$ and the hyperplanes of the corresponding projected arrangement in $\mathbb{R}$, we define the hyperplanes as $\hat{h}_i$, for all $i = 1, \ldots, n$. Recall that the hyperplanes $\hat{h}_i$ are defined as $\hat{h}_i = \{\lambda \in \mathbb{R}^2 : \langle p_i^\bullet, \lambda \rangle = 0\}$. The root weight $\lambda_{2,i}^0$ and the corresponding weight $\lambda_{1,i}^0 = 1 - \lambda_{2,i}^0$ are the coordinates of the intersection point of the arrangement with the normalized weight space $\widetilde{\mathcal{W}}^0$:

$$\left\langle \begin{pmatrix} p_i^1 \\ p_i^2 \end{pmatrix}, \begin{pmatrix} \lambda_{1,i}^0 \\ \lambda_{2,i}^0 \end{pmatrix} \right\rangle = p_i^1 \cdot \left(1 - \frac{-p_i^1}{p_i^2 - p_i^1}\right) + p_i^2 \cdot \frac{-p_i^1}{p_i^2 - p_i^1} = 0.$$

If $p_i^1 = p_i^2$, then the corresponding hyperplane $\hat{h}_i$ is the bisector of the second and fourth quadrant, which is parallel to the normalized weight space $\widetilde{\mathcal{W}}^0$. This is in accordance to the fact that $\lambda_{2,i}^0$ is not defined in this case.

Coming back to the projected arrangement of hyperplanes, the introduction of the half-spaces h_i^- and h_i^+ allows to compute all supported solutions of (2o.0c): For $\lambda_2 \in \mathcal{W}^0$, we transfer the results of Corollary 3.12:

$$x_i \begin{cases} = 0 & \text{if } \mathrm{Pos}_i(\lambda_2) = -1 \\ \in \{0, 1\} & \text{if } \mathrm{Pos}_i(\lambda_2) = 0 \\ = 1 & \text{if } \mathrm{Pos}_i(\lambda_2) = +1 \end{cases}$$

for $i = 1, \ldots, n$.

Every cell $\varphi^{(1)}$ (1-face, edge) of the arrangement of hyperplanes $\{h_1, \ldots, h_n\}$ in $\mathbb{R}$, that intersects $\mathcal{W}^0$, corresponds to one unique extreme supported solution of (2o.0c) (cf. Corollary 3.7 and intervals I_t introduced after Corollary 3.12) and all extreme supported solutions of (2o.0c) are represented by cells $\varphi^{(1)}$ of the arrangement. Thus, we can state:

Corollary 3.15 *The bi-objective unconstrained combinatorial optimization problem has at most $\mathcal{O}(n)$ extreme supported efficient solutions.*

If the arrangement of hyperplanes is simple, no nonextreme supported solutions occur. As described in Section 3.1.4, the position vectors of the facets $\varphi^{(0)}$, the subfaces of the cells, differ in one single position, where the entry is 0. Hence, a facet $\varphi^{(0)}$ corresponds to two supported solutions that are also induced by the adjacent cells.

If the arrangement of hyperplanes is not simple, there might be an exponential number of nonextreme supported solutions. If, for example, all hyperplanes h_i are identical, there

are two extreme and $(2^n - 2)$ nonextreme supported solutions for $\lambda^0_{2,i} \in \mathcal{W}^0$. The number of corresponding nonextreme supported nondominated points can be smaller if there are equivalent solutions. Solutions x and $\bar{x}$ are called *equivalent* if $f(x) = f(\bar{x})$. Ehrgott [2005] presented an instance with 2^n feasible, non-equivalent solutions that all correspond to supported points, where only two of them are extreme:

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} -2^{i-1} x_i \\ \max \quad & \sum_{i=1}^{n} 2^{i-1} x_i \\ \text{s.t.} \quad & x_i \in \{0,1\}, \qquad i = 1, \ldots, n. \end{aligned}$$

For this instance we get $\lambda^0_{2,i} = 1/2$ for all items $i = 1, \ldots, n$. Thus, the arrangement of hyperplanes consists of two cells and one facet. Nevertheless, also in the case of non-simple arrangements, all nonextreme solutions can be generated by going through the $\mathcal{O}(n)$ facets (0-faces, vertices) and building all possible assignments for items i with entries of the position vector $\mathrm{Pos}_i(\lambda_2)$ equal to 0.

3.3.2 Multi-objective unconstrained combinatorial optimization problems

The set of supported solutions $\mathcal{X}_{sE}$ of the multi-objective unconstrained combinatorial optimization problem (mo.0c) for three and more objective functions can be computed in a similar way as for the bi-objective problem. As above, for each item i, for $i = 1, \ldots, n$, the weighted utility $\rho_i : \mathbb{R}^{m-1} \to \mathbb{R}$

$$\rho_i(\lambda_2, \ldots, \lambda_m) = (1 - \sum_{j=2}^{m} \lambda_j) \cdot p_i^1 + \sum_{j=2}^{m} \lambda_j \cdot p_i^j$$

defines a hyperplane

$$h_i = \left\{ (\lambda_2, \ldots, \lambda_m) \in \mathbb{R}^{m-1} \,:\, \rho_i(\lambda_2, \ldots, \lambda_m) = 0 \right\}$$

and two half-spaces

$$\begin{aligned} h_i^- &= \left\{ (\lambda_2, \ldots, \lambda_m) \in \mathbb{R}^{m-1} \,:\, \rho_i(\lambda_2, \ldots, \lambda_m) < 0 \right\} \quad \text{and} \\ h_i^+ &= \left\{ (\lambda_2, \ldots, \lambda_m) \in \mathbb{R}^{m-1} \,:\, \rho_i(\lambda_2, \ldots, \lambda_m) > 0 \right\} \end{aligned}$$

in $\mathbb{R}^{m-1}$.

Remark 3.16 *Let $i \in \{1, \ldots, n\}$. If $p_i^j \neq p_i^1$ for all $j \in \{2, \ldots, m\}$, then we can equivalently define the hyperplane h_i as the affine hull of the $(m-1)$ affinely independent* root weights

$$\lambda^0_{j,i} = \frac{-p_i^1}{p_i^j - p_i^1} \cdot e_{j-1}, \qquad j = 2, \ldots, m,$$

where e_{j-1} is the $(j-1)$-th unit vector in $\mathbb{R}^{m-1}$. Note that these points resemble the root weights of Definition 3.10. The arrangement of hyperplanes intersecting with a certain axis in $\mathbb{R}^{m-1}$ corresponds to a bi-objective problem where the first and j-th objective are considered, since λ_j and $\lambda_1 = 1-\lambda_j$ are the only weights not equal to 0. The corresponding root weights define the hyperplane h_i.

If $p_i^j = p_i^1$ for some $j \in \{2, \dots, m\}$, the hyperplane runs parallel to the $(j-1)$-th axis. Furthermore, if $p_i^j = p_i^1$ for all $j \in \{2, \dots, m\}$, then h_i is empty and either $h_i^- = \mathbb{R}^{m-1}$ and $h_i^+ = \emptyset$ or $h_i^- = \emptyset$ and $h_i^+ = \mathbb{R}^{m-1}$, similar to the bi-objective case.

The faces of the arrangement of hyperplanes $\{h_1, \dots, h_n\}$ in $\mathbb{R}^{m-1}$ that intersect with the projected weight space $\mathcal{W}^0$ correspond to supported solutions of (mo.0c). All extreme supported solutions can be identified by the cells $\varphi^{(m-1)}$ of the arrangement (cf. Corollary 3.7). Hence, we can state:

Corollary 3.17 *The multi-objective unconstrained combinatorial optimization problem has at most $\mathcal{O}(n^{m-1})$ extreme supported efficient solutions.*

As described in Section 3.2.3, the duality of the arrangement of hyperplanes in the projected weight space $\mathcal{W}^0$ and the convex hull of the set of nondominated points $\text{conv}(\mathcal{Y}_N)$ in the objective space has an order reversing characteristic: All weights of a k-face in $\mathcal{W}^0 \subset \mathbb{R}^{m-1}$ correspond to an $((m-1)-k)$-face in $\text{conv}(\mathcal{Y}_N) \subset \mathbb{R}^m$, thus

$$\text{weights of a } \left\{ \begin{array}{c} \text{cell} \\ \text{facet} \\ \vdots \\ \text{vertex} \end{array} \right\} \text{ in } \mathcal{W}^0 \text{ correspond to a } \left\{ \begin{array}{c} \text{vertex} \\ \text{edge} \\ \vdots \\ \text{facet} \end{array} \right\} \text{ in } \text{conv}(\mathcal{Y}_N).$$

A cell $\varphi^{(m-1)}$ in $\mathcal{W}^0$ always corresponds to one unique supported point since the position vector has no 0-entries. In the case of a simple arrangement, it can be stated more precisely that a facet $\varphi^{(m-2)}$ in $\mathcal{W}^0$ corresponds to two extreme supported points. The position vector has exactly one 0-entry and, therefore, indicates that the weighted utility $\rho_i(\lambda_2, \dots, \lambda_m) = 0$ for exactly one item $i \in \{1, \dots, n\}$ for weights $(\lambda_2, \dots, \lambda_m) \in \varphi^{(m-2)}$. The corresponding weighted sum objective is undecided on item i. In general, a k-face $\varphi^{(k)}$, $m-1 \geq k \geq 0$, has $((m-1)-k)$ 0-entries in the position vector and, hence, corresponds to $2^{(m-1)-k}$ extreme supported points and extreme supported solutions, respectively.

If the arrangement is not simple, $\varphi^{(k)}$ may have a position vector with more than $((m-1)-k)$ 0-entries and the number of corresponding supported solutions may change since nonextreme supported solutions occur. For computing all nonextreme supported solutions it is sufficient to analyze only k-faces, $0 \leq k < m-1$, that have no subfaces. Recall that all 0-entries are passed on to subfaces and therefore no information is lost by skipping faces that have subfaces.

3.3.3 Case study: tri-objective unconstrained combinatorial optimization problems with one positive and two negative objective functions

In this section, we illustrate the above results on tri-objective unconstrained combinatorial optimization problems:

$$\begin{aligned} \text{vmax} \quad & f(x) = \left(\sum_{i=1}^{n} p_i^1 x_i \, , \, \sum_{i=1}^{n} p_i^2 x_i \, , \, \sum_{i=1}^{n} p_i^3 x_i \right) \\ \text{s.t.} \quad & x_i \in \{0,1\}, \qquad i = 1, \ldots, n. \end{aligned} \tag{3o$_{pnn}$.0c}$$

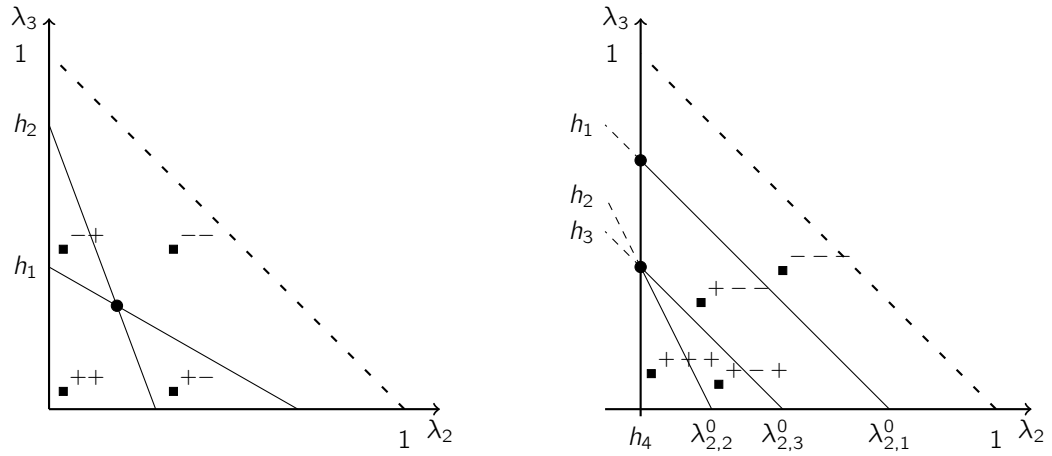
We assume a special structure, namely that all coefficients of the first objective function are positive and all coefficients of the second and third objective function are negative. More precisely, $p_i^1 > 0$, $p_i^2 < 0$, and $p_i^3 < 0$, for all $i = 1, \ldots, n$.

We know that for problem (3o$_{pnn}$.0c) every hyperplane h_i, for $i = 1, \ldots, n$, intersects with the λ_2- and λ_3-axis at the root weights $\lambda^0_{2,i}$ and $\lambda^0_{3,i}$, where the intercept of the axis is in the interval $[0, 1]$, c. f. Type (2) of Lemma 3.11 for (2o.0c). Hence, every hyperplane intersects with the projected weight space $\mathcal{W}^0 = \{(\lambda_2, \lambda_3) \in \mathbb{R}^2_> \, : \, \lambda_2 + \lambda_3 \leq 1\}$. Furthermore, every half-space h_i^- lies *above* and every half-space h_i^+ lies *below* the corresponding hyperplane h_i, i. e., given a point $(\bar{\lambda}_2, \bar{\lambda}_3) \in h_i$, it holds that all points $(\bar{\lambda}_2, \lambda_3)$ with $\lambda_3 > \bar{\lambda}_3$ are in h_i^- and that all points $(\bar{\lambda}_2, \lambda_3)$ with $\lambda_3 < \bar{\lambda}_3$ are in h_i^+. Analogously, we denote faces to lie above or below a given hyperplane if they are subsets of h_i^- and h_i^+, respectively. Hence, the top-most cell of the arrangement that lies above all hyperplanes has a position vector with only entries equal to (-1). This implies that the solution $x_i = 0$ for all $i = 1, \ldots, n$ is supported. This agrees with the fact that for two of the objective functions the objective function value decreases if any item is included.

Motivated by the bi-objective case, and in contrast to Ferrez et al. [2005] (where the solution approach is based on visiting every cell of the arrangement, see also Section 3.1.4), we focus the search for supported solutions on the intersection points of hyperplanes, i. e., on the vertices of the arrangement. This is possible since all information is passed from the adjacent faces to the vertices, except for the case that a hyperplane does not intersect any other hyperplane in the projected weight space, see Figure 3.8(b). We first discuss the general case, that is, the intersection of two hyperplanes, and continue with the special case later on. For ease of readability, we introduce the terms *above* and *below* also to describe the relative positions of vertices and two of its adjacent cells. The cell that is adjacent to the intersection point is called to be *lying above/below the intersection point* if it is lying above/below all hyperplanes that define this vertex.

The intersection point $\lambda = (\lambda_2, \lambda_3)$ of the two hyperplanes h_i and h_j, for $i, j \in \{1, \ldots, n\}$ with $\lambda \in \mathcal{W}^0$, is a vertex of the arrangement of hyperplanes and has a position vector with $\text{Pos}_i(\lambda) = 0 = \text{Pos}_j(\lambda)$. The vertex λ is adjacent to four cells that have the four possible combinations of values (-1) and $(+1)$ as the respective entries of the position vector: The solution with $x_i = x_j = 0$ corresponds to the cell above the intersection point, the solution with $x_i = x_j = 1$ corresponds to the cell below the intersection point, and the mixed assignments correspond to the respective cells that are below one and above the

other of the two hyperplanes, cf. Figure 3.8(a). All remaining entries of the position vectors are equal for the vertex and the adjacent cells. Thus, the corresponding supported solutions can be generated by identifying the indices of the intersecting hyperplanes and the position of the vertex with respect to all other hyperplanes.



(a) General Case: intersection of two hyperplanes.

(b) Intersections of hyperplanes and the λ_3-axis.

Figure 3.8: Arrangements of hyperplanes in $\mathcal{W}^0$ associated to $(3o_{pnn}.0c)$. The strings in $\{-,+\}^2$ refer to the position vectors of corresponding cells (symbol $-$ for entry (-1) and symbol $+$ for entry $(+1)$).

The special case of hyperplanes that do not intersect with other hyperplanes in the projected weight space $\mathcal{W}^0$ can be handled in the following way: We enlarge the projected weight space and include weights λ with $\lambda_2 = 0$. Furthermore, we introduce the λ_3-axis as additional hyperplane h_{n+1}. Every hyperplane h_i, for $i = 1, \ldots, n$, intersects h_{n+1}, especially the ones that do not intersect any other hyperplane. Thus, the information of cells that would have no adjacent vertices is passed to the vertices in the λ_3-axis. In return, these vertices may also include information about cells that do not intersect the projected weight space. This case occurs if two or more hyperplanes intersect h_{n+1}, see Figure 3.8(b). Due to the fixed signs in the objective functions we know that all hyperplanes have a negative slope and that the corresponding negative half-spaces are all lying above the hyperplane. Assume that t hyperplanes intersect h_{n+1} in the same vertex. Hence, $(t+1)$ of the adjacent cells intersect the projected weight space. We look at the corresponding solutions: Including none of the t items corresponds to the topmost cell and including all of them to the lowermost. From the topmost to the lowermost cell, one entry of the position vector is switched from (-1) to $(+1)$ by passing one of the t hyperplanes. In fact, the exact ordering can be identified by the decreasing order of the intersection points $\lambda^0_{2,i}$ of the hyperplanes with the λ_2-axis.

Further special cases can occur if the arrangement of hyperplanes is non-simple: Vertices may be adjacent to more than four cells and nonextreme supported solutions have to be

considered. Since the projected weight space is in $\mathbb{R}^2$, the number of different cases is limited:

Parallel hyperplanes do not induce nonextreme supported solutions.

***t* identical hyperplanes** occur if t vectors of coefficients $p_i^\bullet = (p_i^1, p_i^2, p_i^3)$, $i \in \{1, \dots, n\}$, are pairwise linearly dependent in $\mathbb{R}^3$. Assume that $p_i^\bullet = \alpha p_{\bar{\imath}}^\bullet$, for $i, \bar{\imath} \in \{1, \dots, n\}$ and for some $\alpha \in \mathbb{R} \setminus \{0\}$. The corresponding hyperplanes h_i and $h_{\bar{\imath}}$ are defined by the root weights $\lambda_{j,i}^0$ and $\lambda_{j,\bar{\imath}}^0$, for $j = 2, 3$, cf. Remark 3.16. It holds

$$\lambda_{j,i}^0 = \frac{-p_i^1}{p_i^j - p_i^1} e_{j-1} = \frac{-\alpha p_{\bar{\imath}}^1}{\alpha p_{\bar{\imath}}^j - \alpha p_{\bar{\imath}}^1} e_{j-1} = \frac{-p_{\bar{\imath}}^1}{p_{\bar{\imath}}^j - p_{\bar{\imath}}^1} e_{j-1} = \lambda_{j,\bar{\imath}}^0, \qquad j = 2, 3.$$

Hence, h_i and $h_{\bar{\imath}}$ are indeed identical, see Figure 3.9(a).

Every edge of the arrangement of hyperplanes on the identical hyperplanes is adjacent to two cells. These cells correspond to the solutions where none or all of the t items are included, respectively. The corresponding edges of the convex hull of the feasible set $\mathrm{conv}(\mathcal{Y}_N)$ in the objective space then consists of two extreme supported points, corresponding to the two cells, and at most $2^t - 2$ nonextreme supported points, corresponding to all possible combinations of including at least one and at most $(t-1)$ of the items. Some of the nonextreme supported solutions can correspond to identical nonextreme supported points if the sum of coefficients for those subsets of items are identical.

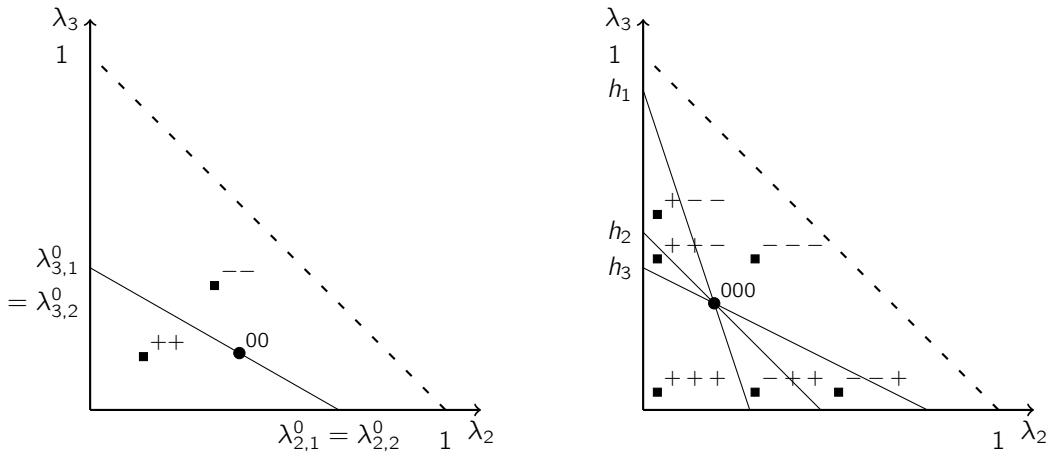
Intersection of *t*, $t > 2$, pairwise distinct hyperplanes in one point occurs if the respective vectors of coefficients $p_i^\bullet = (p_i^1, p_i^2, p_i^3)$, $i \in \{1, \dots, n\}$, are linearly dependent, but pairwise linearly independent in $\mathbb{R}^3$. Assume that $p_i^\bullet = \alpha p_{\bar{\imath}}^\bullet + \beta p_{\hat{\imath}}^\bullet$, for $i, \bar{\imath}, \hat{\imath} \in \{1, \dots, n\}$ and for some $\alpha, \beta \in \mathbb{R} \setminus \{0\}$. The weighted utility $\rho_i(\lambda_2, \lambda_3)$ of item i is equal to:

$$\begin{aligned}
\rho_i(\lambda_2, \lambda_3) &= (1 - \lambda_2 - \lambda_3) p_i^1 + \lambda_2 p_i^2 + \lambda_3 p_i^3 \\
&= (1 - \lambda_2 - \lambda_3)(\alpha p_{\bar{\imath}}^1 + \beta p_{\hat{\imath}}^1) + \lambda_2 (\alpha p_{\bar{\imath}}^2 + \beta p_{\hat{\imath}}^2) + \lambda_3 (\alpha p_{\bar{\imath}}^3 + \beta p_{\hat{\imath}}^3) \\
&= \alpha \big((1 - \lambda_2 - \lambda_3) p_{\bar{\imath}}^1 + \lambda_2 p_{\bar{\imath}}^2 + \lambda_3 p_{\bar{\imath}}^3 \big) + \beta \big((1 - \lambda_2 - \lambda_3) p_{\hat{\imath}}^1 + \lambda_2 p_{\hat{\imath}}^2 + \lambda_3 p_{\hat{\imath}}^3 \big) \\
&= \alpha \rho_{\bar{\imath}}(\lambda_2, \lambda_3) + \beta \rho_{\hat{\imath}}(\lambda_2, \lambda_3).
\end{aligned}$$

It holds:

$$\begin{aligned}
h_i \cap h_{\bar{\imath}} &= \big\{ (\lambda_2, \lambda_3) \in \mathbb{R}^2 \,:\, \rho_i(\lambda_2, \lambda_3) = 0 \,\wedge\, \rho_{\bar{\imath}}(\lambda_2, \lambda_3) = 0 \big\} \\
&= \big\{ (\lambda_2, \lambda_3) \in \mathbb{R}^2 \,:\, \alpha \rho_{\bar{\imath}}(\lambda_2, \lambda_3) + \beta \rho_{\hat{\imath}}(\lambda_2, \lambda_3) = 0 \,\wedge\, \rho_{\bar{\imath}}(\lambda_2, \lambda_3) = 0 \big\} \\
&= \big\{ (\lambda_2, \lambda_3) \in \mathbb{R}^2 \,:\, \rho_{\hat{\imath}}(\lambda_2, \lambda_3) = 0 \,\wedge\, \rho_{\bar{\imath}}(\lambda_2, \lambda_3) = 0 \big\} \\
&= h_{\hat{\imath}} \cap h_{\bar{\imath}}
\end{aligned}$$

Hence, the hyperplanes h_i, $h_{\bar{\imath}}$ and $h_{\hat{\imath}}$ intersect in one unique point.



(a) Two identical hyperplanes in $\mathcal{W}^0$.

(b) Intersection of three hyperplanes in one unique point.

Figure 3.9: Arrangements of hyperplanes in $\mathcal{W}^0$ associated to $(3o_{pnn}.0c)$. The strings in $\{-,0,+\}^2$ and $\{-,0,+\}^3$, respectively, refer to the position vectors of corresponding faces (symbol $-$ for entry (-1) and symbol $+$ for entry $(+1)$).

If two hyperplanes intersect, the corresponding vertex is adjacent to four cells. Every additional, but distinct hyperplane that intersects the same vertex splits two of these cells, cf. Figure 3.9(b). Thus, a vertex defined by t, $t > 2$, pairwise distinct hyperplanes is adjacent to $2t$ cells. The cell above the intersection point again corresponds to the solution where none of the t items are selected and the cell below the intersection point corresponds to the solution where all of the t items are selected. Visiting all adjacent cells in counterclockwise direction around the intersection point, starting at the cell above, at each iteration one hyperplane is passed and, thus, one variable value is switched from 0 to 1. After t iterations all of the t items are included, which corresponds to the cell below the intersection point. Every hyperplane has been passed from the top-most to the bottom-most, left-hand of the intersection point and, continuing, is passed again and in the same order from the bottom-most to the top-most, right-hand of the intersection point. Hence, iteratively and in the same order, the variable values are switched back from 1 to 0 until, after t iterations, the cell above the intersection point is reached again. Therefore, the intersection point in the projected weight space corresponds to a face of the convex hull $\text{conv}(\mathcal{Y}_N)$, which includes the $2t$ extreme supported points corresponding to these solutions on the boundary of the face and $2^t - 2t$ nonextreme supported points in the interior of the face.

The combination of both cases induces faces in the objective space with nonextreme supported points in the interior and on some edges of the face.

Remark 3.18 *The structure of the arrangement of hyperplanes allows to determine the*

exact number of extreme supported points that define a facet of the convex hull $\text{conv}(\mathcal{Y})$*, which is even in any case, and the exact number of nonextreme supported solutions.*

Example 3.19 *Consider the following instance of* $(3o_{pnn}.0c)$*. We determine the set of supported solutions* $\mathcal{X}_{sE}$*.*

$$\begin{array}{lll} \max & 16x_1 + 21x_2 + 10x_3 + 9x_4 + 3x_5 \\ \max & -24x_1 - 14x_2 - 10x_3 - x_4 - 27x_5 \\ \max & -4x_1 - 9x_2 - 10x_3 - 21x_4 - 12x_5 \\ \text{s.t.} & x_i \in \{0,1\}, \quad i = 1,\dots,5. \end{array}$$

The hyperplanes h_i *corresponding to items* i*,* $i = 1,\dots,5$*, are visualized in Figure 3.10. They yield ten cells in* $\mathcal{W}^0$*, hence, the problem has ten extreme supported solutions. The arrangement is not simple since three hyperplanes,* h_1*,* h_3 *and* h_4*, intersect in one point. Therefore, also nonextreme supported solutions occur. Table 3.3 presents all* 32 *feasible solutions, the corresponding objective function values and indicates if these solutions are efficient (*17 *solutions), supported efficient (*12 *solutions) or extreme supported efficient (*10 *solutions).*

There are only three vertices of the arrangement inside the projected weight space. They inherit the information about nine of the ten cells. Hyperplane h_5 *does not intersect any other hyperplane in* $\mathcal{W}^0$*. The solution* $x = (1,1,1,1,1)^\top$ *can be determined by considering the intersection of* h_5 *with the* λ_3*-axis.*

The algorithm Our solution algorithm for $(3o_{pnn}.0c)$ is based on the work of Bentley and Ottmann [1979] for reporting the intersections of line segments in the plane. Adopted to our problem, the line segments are defined as the intersections of the respective hyperplanes and the projected weight space. The algorithm sweeps with a vertical line from left to right through the projected weight space, starting at $\lambda_2 = 0$ and stopping at $\lambda_2 = 1$ (cf. Figure 3.11). Every vertical line defines a permutation $\pi \in \mathcal{S}_n$ of the numbers $1,\dots,n$, where $\mathcal{S}_n$ denotes the symmetric group of order n. Given a fixed weight λ_2, the respective points (λ_2, λ_3) on the hyperplanes $h_1,\dots,h_n$ define a permutation of indices $1,\dots,n$ in non-increasing order of the weights λ_3. This permutation is unique if and only if every value λ_3 corresponding to one hyperplane is unique. Two or more hyperplanes define equal λ_3-values if the hyperplanes intersect. Multiple permutations correspond to this vertical line.

The idea of the method of Bentley and Ottmann [1979] is that, sweeping the vertical line from left to right, the permutation only changes if two or more line segments intersect at the corresponding λ_2-value and that in this case only hyperplanes that are subsequent in the permutation can intersect. Hence, starting from $\lambda_2 = 0$, the permutation π is initialized and the intersections of subsequent hyperplanes in the permutation are computed. All intersection points are stored in a list Λ. In non-increasing order of λ_2, the algorithm iteratively sweeps the vertical line through the λ_2-components of the points in Λ. An intersection of hyperplanes induces that their positions switch at that point, see Figure 3.11.

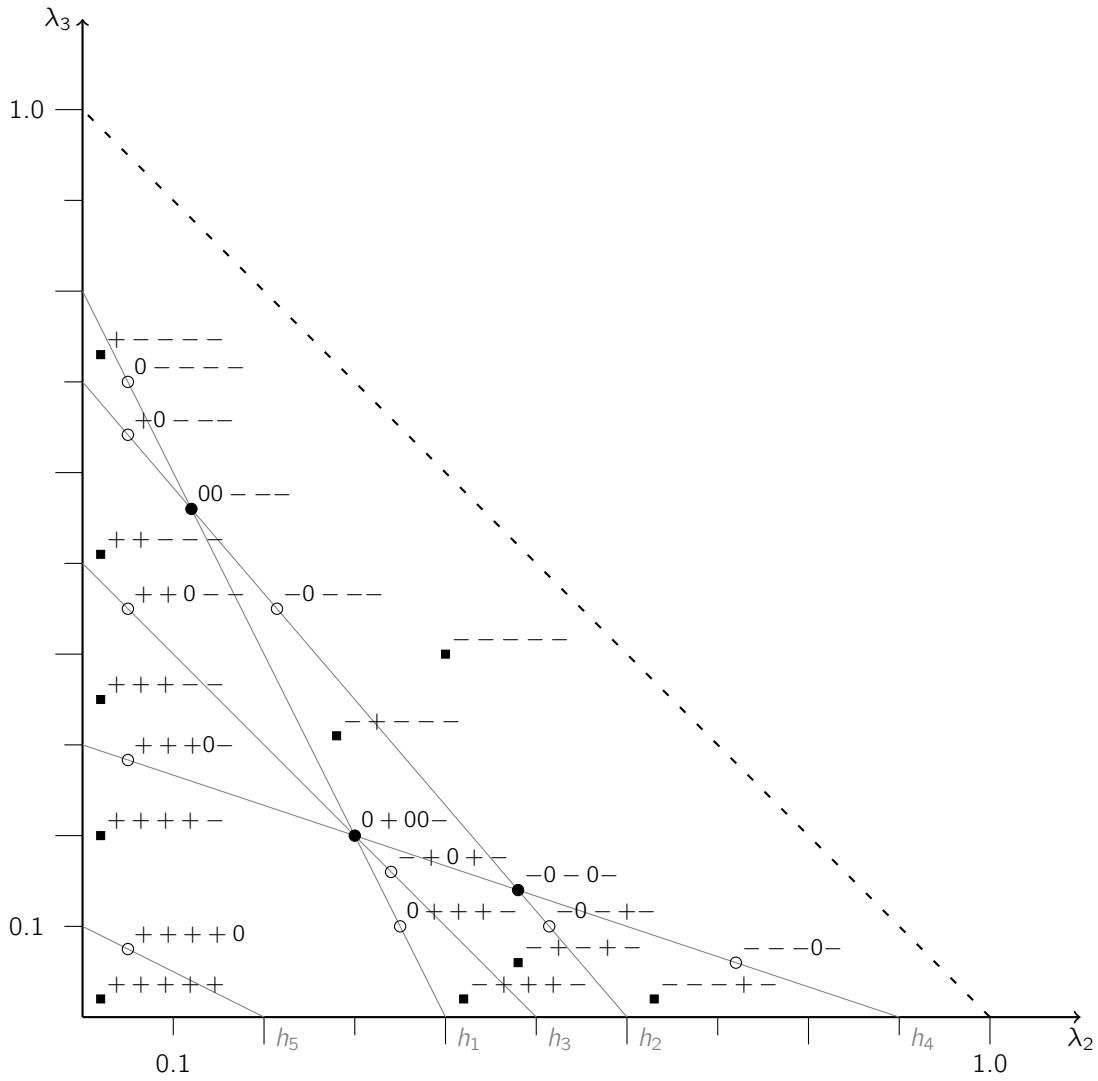


Figure 3.10: Projected weight space $\mathcal{W}^0$ with arrangement of hyperplanes $\{h_1, \ldots, h_5\}$ for Example 3.19. The strings in $\{-, 0, +\}^5$ refer to the position vectors $\mathrm{Pos}_i((\lambda_2, \lambda_3))$ with symbol $-$ for entry (-1), symbol 0 for entry 0, and symbol $+$ for entry $(+1)$. The symbols ▪ identify a point of a cell, the symbols ○ identify a point of a facet, and the symbols ● identify vertices of the arrangement, respectively.

x_1	x_2	x_3	x_4	x_5	$f_1(x)$	$f_2(x)$	$f_3(x)$	efficient	supported	extreme
0	0	0	0	0	0	0	0	×	×	×
1	0	0	0	0	16	-24	-4	×	×	×
0	1	0	0	0	21	-14	-9	×	×	×
0	0	1	0	0	10	-10	-10	×		
0	0	0	1	0	9	-1	-21	×	×	×
0	0	0	0	1	3	-27	-12			
1	1	0	0	0	37	-38	-13	×	×	×
1	0	1	0	0	26	-34	-14	×		
1	0	0	1	0	25	-25	-25			
1	0	0	0	1	19	-51	-16			
0	1	1	0	0	31	-24	-19	×	×	
0	1	0	1	0	30	-15	-30	×	×	×
0	1	0	0	1	24	-41	-21			
0	0	1	1	0	19	-11	-31	×		
0	0	1	0	1	13	-37	-22			
0	0	0	1	1	12	-28	-33			
1	1	1	0	0	47	-48	-23	×	×	×
1	1	0	1	0	46	-39	-34	×	×	
1	1	0	0	1	40	-65	-25			
1	0	1	1	0	35	-35	-35	×		
1	0	1	0	1	29	-61	-26			
1	0	0	1	1	28	-52	-37			
0	1	1	1	0	40	-25	-40	×	×	×
0	1	1	0	1	34	-51	-31			
0	1	0	1	1	33	-42	-42			
0	0	1	1	1	22	-38	-43			
1	1	1	1	0	56	-49	-44	×	×	×
1	1	1	0	1	50	-75	-35	×		
1	1	0	1	1	49	-66	-46			
1	0	1	1	1	38	-62	-47			
0	1	1	1	1	43	-52	-52			
1	1	1	1	1	59	-76	-56	×	×	×
							$\sum$	17	12	10

Table 3.3: Feasible solutions, corresponding objective function values and dominance characteristic for Example 3.19.

The permutation can easily be updated at every iteration. Due to this update, the intersections of newly subsequent hyperplanes have to be computed and, if the λ_2-component is larger than the current λ_2-value of the vertical line, i. e., if the intersection point lies right hand of the vertical line, added to Λ. The algorithm stops if the list of intersection points is empty. All intersection points of the hyperplanes in the projected weight space have been computed during the algorithm.

Example 3.20 *Look at the example in Figure 3.11. Four hyperplanes are intersecting in the projected weight space. The algorithm of Bentley and Ottmann [1979] proceeds as follows:*

Vertical line	π	*Intersections*	Λ
$\lambda_2 = 0$	$(1,2,3,4)$	$h_1 \cap h_2 \cap \mathcal{W}^0 = \{(0.2, 0.3)\}$ $h_2 \cap h_3 \cap \mathcal{W}^0 = \emptyset$ $h_3 \cap h_4 \cap \mathcal{W}^0 = \{(0.1, 0.2)\}$	$\{(0.1, 0.2), (0.2, 0.3)\}$
$\lambda_2 = 0.1$	$(1,2,4,3)$	$h_2 \cap h_4 \cap \mathcal{W}^0 = \{(0.4, 0.1)\}$	$\{(0.2, 0.3), (0.4, 0.1)\}$
$\lambda_2 = 0.2$	$(2,1,4,3)$	$h_1 \cap h_4 \cap \mathcal{W}^0 = \{(0.25, 0.15)\}$	$\{(0.25, 0.15), (0.4, 0.1)\}$
$\lambda_2 = 0.25$	$(2,4,1,3)$	$h_2 \cap h_4 \cap \mathcal{W}^0 = \{(0.4, 0.1)\}$ $h_1 \cap h_3 \cap \mathcal{W}^0 = \emptyset$	$\{(0.4, 0.1)\}$
$\lambda_2 = 0.4$	$(4,2,1,3)$	$h_2 \cap h_1 \cap \mathcal{W}^0 = \{(0.2, 0.3)\}$	$\emptyset$

The algorithm initializes with the vertical line at $\lambda_2 = 0$. The first permutation $\pi = (1,2,3,4)$ induces that the intersections of h_1 and h_2, h_2 and h_3, and h_3 and h_4 and the projected weight space have to be computed. Two intersection points are added to Λ and the vertical line is swept to $\lambda_2 = 0.1$. Continuing, the permutation π for $\lambda_2 + \varepsilon$, for a small $\varepsilon > 0$, the intersection of newly subsequent hyperplanes and the resulting set of intersections Λ is given. The algorithm of Bentley and Ottmann [1979] computes all intersection points in five iterations.

The algorithm of Bentley and Ottmann [1979] has a complexity of $\mathcal{O}(n \log n + k \log n)$, where k is the number of intersection points, which is in $\mathcal{O}(n^2)$ for an arrangement of hyperplanes in $\mathbb{R}^2$. Bentley and Ottmann [1979] already indicated that the naïve approach, of testing all $\binom{n}{2}$ possible intersections of pairs of hyperplanes, with a complexity of $\mathcal{O}(n^2)$ becomes more efficient than their approach if k is very close to n^2. Nevertheless, we prefer the algorithm of Bentley and Ottmann [1979] since it is very unlikely that nearly all intersection points of the arrangement of hyperplanes are inside the projected weight space $\mathcal{W}^0$.

Now that we know how to compute all intersection points in the projected weight space, i. e., the vertices of the arrangement of hyperplanes, the set of extreme supported solutions of (3o$_{pnn}$.0c) can be easily generated. In Algorithm 3.2 a possible procedure is presented. The list Λ consists of entries $(\lambda_2, \lambda_3, j, t)$ that describe intersection points of the $(t+1)$ hyperplanes $h_{\pi(j)}$ to $h_{\pi(j+t)}$ at the point (λ_2, λ_3). In the general case two hyperplanes

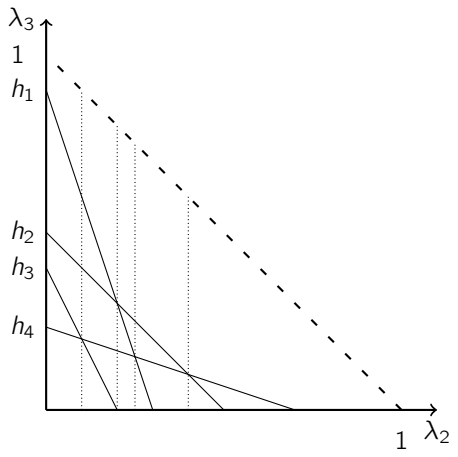


Figure 3.11: The method of Bentley and Ottmann [1979] sweeps with a vertical line through the projected weight space to identify all intersection points.

define one intersection point and, hence, $t = 1$. The list Λ is sorted and the k-th entry of Λ is identified by squared brackets: $\Lambda[k]$. If $\Lambda[1]$ is deleted from Λ, all following entries increase one position in rank. If a new entry is inserted at $\Lambda[k]$, the current entry $\Lambda[k]$ and all following entries scale down one position. The last entry $\Lambda[\text{end}]$ is assumed to be an empty set.

At each iteration of the algorithm, i. e., for each intersection point, the corresponding solutions are evaluated. Since we know which hyperplanes are intersecting in the current intersection point, the permutation π of the corresponding vertical line includes all necessary information. For clarity, we use the index i to identify hyperplanes and the index j to identify the position of the hyperplanes in π. All hyperplanes $h_{\pi(\hat{\jmath})}$ that have a position $\hat{\jmath}$ in π that is smaller than the position j of the top-most intersecting hyperplane, are lying above the intersection point. Hence, the intersection point is located in the corresponding half-spaces $h^+_{\hat{\jmath}}$ and the variables $x_{\hat{\jmath}}$ have to be set to 1. Vice versa, all hyperplanes $h_{\hat{\jmath}}$ that have a position $\hat{\jmath}$ in π that is larger than the position $(j + t)$ of the lower-most intersecting hyperplane, are lying below the intersection point. Hence, the intersection point is located in the corresponding half-spaces $h^-_{\hat{\jmath}}$ and the variables $x_{\hat{\jmath}}$ have to be set to 0. The $(t + 1)$ intersecting hyperplanes define the remaining $(t + 1)$ variables.

All special cases are handled in Algorithm 3.2. At first, identical hyperplanes are filtered out. Assume that the hyperplanes $h_{\hat{\imath}}$ and $h_{\bar{\imath}}$ are identical. Again, due to the fixed signs of the profits $p^{\bullet}_{\hat{\imath}}$ and $p^{\bullet}_{\bar{\imath}}$, it holds that $h^+_{\hat{\imath}} = h^+_{\bar{\imath}}$ and $h^-_{\hat{\imath}} = h^-_{\bar{\imath}}$. Thus, for every weight $(\lambda_2, \lambda_3) \in \mathcal{W}^0$, it holds for the corresponding optimal solution that $x_{\hat{\imath}} = x_{\bar{\imath}}$. Hence, by storing the information that the hyperplanes are identical, the hyperplane $h_{\bar{\imath}}$ can be filtered out to simplify the procedure (function FilterIdentical) and the solutions are completed at the end of the algorithm (function ReconstructIdentical).

Continuing with all pairwise distinct hyperplanes, Algorithm 3.2 identifies the permutation π corresponding to the vertical line with $\lambda_2 = 0$. The special case of vertices on the

Algorithm 3.2 Generates all extreme supported solutions of (3o$_{pnn}$.0c).

Input: profit vectors $p_i^\bullet$, $i \in \{1, \ldots, \tilde{n}\}$

1: generate the hyperplanes h_i defined by the profit vectors $p_i^\bullet$, for $i = 1, \ldots, \tilde{n}$
2: $(h_1, \ldots, h_n, a) :=$FilterIdentical$(h_1, \ldots, h_{\tilde{n}})$
3: global list $\Lambda := \Lambda[1] = \emptyset$ *// list of intersection points and corresponding hyperplanes*
4: $x := \mathbf{0}_n$ *// top-most cell*
5: global set $\mathcal{X}_{eE} := \{x\}$
6: let $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ be the permutation of items $i = \pi(j)$ such that

$$\begin{cases} \lambda_{3,\pi(j)} > \lambda_{3,\pi(j+1)}, \text{ or} \\ \lambda_{3,\pi(j)} = \lambda_{3,\pi(j+1)} \text{ and } \lambda_{2,\pi(j)} > \lambda_{2,\pi(j+1)} \end{cases}$$

for $j = 1, \ldots, n-1$ with $\lambda_{k,\pi(j)} = \frac{-p_i^1}{p_i^k - p_i^1}$ for $k = 2, 3$.
7: **for** $j := 1, \ldots, n$ **do** *// cells adjacent to the λ_3-axis*
8: $x_{\pi(j)} := 1$, $\mathcal{X}_{eE} := \mathcal{X}_{eE} \cup \{x\}$
9: **if** $j < n$ **then**
10: ComputeIntersection$(h_{\pi(j)}, h_{\pi(j+1)}, 0)$
11: **while** $\Lambda[1] \neq \emptyset$ **do** *// sweep through intersection points*
12: $(\bar{\lambda}_2, \bar{\lambda}_3, j, t) := \Lambda[1]$
13: $\Lambda := \Lambda[2 : \text{end}]$ *// delete $\Lambda[1]$ from list Λ*
14: GenerateSolutions(π, j, t) *// cells adjacent to intersection point*
15: $\pi :=$UpdatePermutation(π, j, t)
16: **if** $j > 1$ **then** *// intersection above*
17: ComputeIntersection$(h_{\pi(j-1)}, h_{\pi(j)}, \bar{\lambda}_2)$
18: **if** $j + t < n$ **then** *// intersection below*
19: ComputeIntersection$(h_{\pi(j+t)}, h_{\pi(j+t+1)}, \bar{\lambda}_2)$
20: ReconstructIdentical$(\tilde{n}, a)$

Output: $\mathcal{X}_{eE}$

Algorithm 3.2 Part 2: Generates all extreme supported solutions of (3o$_{pnn}$.0c).

function FILTERIDENTICAL($g_1, \ldots, g_{\tilde{n}}$) *// merge identical hyperplanes*
  $n := 0$, $a := \mathbf{0}_{\tilde{n}}$
  for $i := 1, \ldots, \tilde{n}$ **do**
    if $a(i) = 0$ **then** *// not identical to previously examined hyperplanes*
      $n := n + 1$, $h_n := g_i$, $a(i) := n$
      for $k := i + 1, \ldots, \tilde{n}$ **do**
        if $g_i = g_k$ **then**
          $a(k) := n$
  return $h_1, \ldots, h_n, a$
end function

procedure COMPUTEINTERSECTION($h_j, h_k, \bar{\lambda}_2$)
  compute intersection point (λ_2, λ_3) of hyperplanes h_j and h_k
  if $(\bar{\lambda}_2 < \lambda_2 < 1)$ **and** $(0 < \lambda_3 < 1)$ **then**
    $r := 1$, stop $:= 0$
    while stop $= 0$ **do**
      if $\Lambda[r] = \emptyset$ **then** *// r = end*
        $\Lambda := \big[\Lambda[1 : r-1], (\lambda_2, \lambda_3, j, 1), \Lambda[\text{end}]\big]$ *// insert at the end of List Λ*
        stop $:= 1$
      else
        $(\hat{\lambda}_2, \hat{\lambda}_3, \hat{\jmath}, \hat{t}) := \Lambda[r]$
        if $(\lambda_2 < \hat{\lambda}_2)$ **or** $(\lambda_2 = \hat{\lambda}_2$ **and** $\lambda_3 > \hat{\lambda}_3)$ **then**
          $\Lambda := \big[\Lambda[1 : r-1], (\lambda_2, \lambda_3, j, 1), \Lambda[r : \text{end}]\big]$ *// insert into List Λ*
          stop $:= 1$
        else
          $r := r + 1$
end procedure

procedure GENERATESOLUTIONS(π, j, t)
  $x := \mathbf{0}_n$
  for $k := 1, \ldots, j-1$ **do** *// hyperplanes above intersection point*
    $x_{\pi(k)} := 1$
  $\mathcal{X}_{eE} := \mathcal{X}_{eE} \cup \{x\}$ *// adjacent cells, proceed counterclockwise*
  for $k := 0, \ldots, t$ **do**
    $x_{\pi(j+k)} := 1$, $\mathcal{X}_{eE} := \mathcal{X}_{eE} \cup \{x\}$
  for $k := 0, \ldots, t-1$ **do**
    $x_{\pi(j+k)} := 0$, $\mathcal{X}_{eE} := \mathcal{X}_{eE} \cup \{x\}$
end procedure

Algorithm 3.2 Part 3: Generates all extreme supported solutions of (3o$_{pnn}$.0c).

```
57: function UpdatePermutation(σ, j, t)
58:     for k := 0, ..., t do                                  // invert ordering
59:         π(j + k) := σ(j + t − k)
60:     return π
61: end function

62: procedure ReconstructIdentical(ñ, a)
63:     X := X_eE, X_eE := ∅, x := 0_ñ
64:     for all x̃ ∈ X do
65:         for i := 1, ..., ñ do
66:             x_i := x̃_a(i)
67:         X_eE := X_eE ∪ {x}
68: end procedure
```

λ_3-axis are taken into account in the definition of the permutation π as described before (Line 6). The algorithm passes from top to bottom through all intersection points of the hyperplanes and the λ_3-axis. For each hyperplane, except the last one, the intersection to the subsequent hyperplane is computed and the solution corresponding to the adjacent cell that is directly below the current hyperplane is generated (Lines 7 to 10).

While there are intersection points in the list Λ, the algorithm generates all solutions of adjacent cells (function GenerateSolutions), updates the permutation π (function UpdatePermutation) and computes new intersection points according to the method of Bentley and Ottmann [1979] (function ComputeIntersection). If an intersection point is defined by more than two hyperplanes, this special case is taken into account for generating the corresponding solutions as described above.

Algorithm 3.2 has a worst case complexity of $\mathcal{O}(n^3 \log n)$. The computation of the intersection points has a worst case complexity of $\mathcal{O}(n^2 \log n)$ (approach of Bentley and Ottmann with $\mathcal{O}(n^2)$ intersection points). For each intersection point the values of $\mathcal{O}(n)$ variables have to be determined. Each solution has to be stored, which is possible in $\mathcal{O}(n)$ time using, for example, a binary search tree. Every level of the tree constitutes the decision of setting one variable x_i, thus, the tree has a depth of n. A new solution can be inserted in $\mathcal{O}(n)$. The same time is needed to detect that a solution has already been stored. Furthermore, the number of leafs of the tree is identical with the number of extreme supported solutions, which is in $\mathcal{O}(n^2)$. Hence, the space complexity of the binary search tree is in $\mathcal{O}(n \cdot n^2) = \mathcal{O}(n^3)$.

Note that Algorithm 3.2 generates a lot of redundant information. A solution is as often generated as the corresponding cell has adjacent vertices. Nevertheless, the effort for computing and saving the solutions is quite small such that it is not reasonable to introduce counteractions.

In case of non-simple arrangements that generate nonextreme supported solutions, the algorithm can be extended to compute all elements of the power set of the intersecting hyperplanes. The generation of solutions has to include solutions that only correspond to the current vertex. This can easily be done by reformulating the function GenerateSolutions (see Algorithm 3.3). To take all nonextreme supported solutions into account, also all possible combinations of items corresponding to identical hyperplanes have to be generated. Hence, identical hyperplanes should not be filtered, the functions FilterIdentical and ReconstructIdentical have to be eliminated for this purpose. The set $\mathcal{X}_{eE}$ has to be replaced by $\mathcal{X}_{sE}$. Including nonextreme supported solutions, the complexity of the variant of Algorithm 3.2 may become exponential, due to the possibly exponential number of supported solutions. However, the incremental complexity, i. e., the time for computing the next supported solution, is still constant.

Algorithm 3.3 Procedure of variant of Algorithm 3.2 for generating all extreme and nonextreme supported solutions of ($3o_{pnn}$.0c).

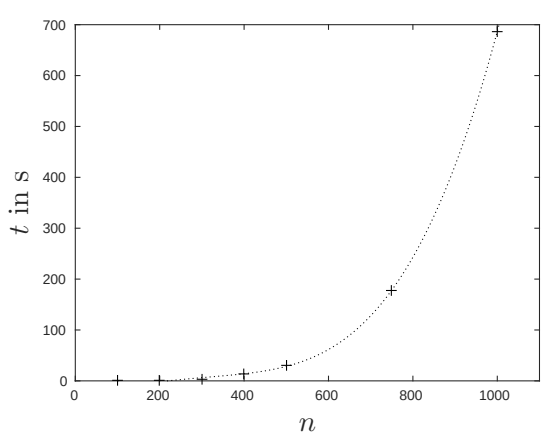
```
1: procedure GenerateSolutions(π, j, t)
2:     x := 0_n
3:     for k := 1, ..., j − 1 do                    // hyperplanes above intersection point
4:         x_π(k) := 1
5:     for all (x_π(j), ..., x_π(j+t)) ∈ {0,1}^{t+1} do    // generate all 2^{t+1} assignments
6:         X_sE := X_sE ∪ {x}
7: end procedure
```

Computational results We implemented the general variant of the algorithm (including nonextreme supported solutions, see Algorithm 3.3) in C++. Experiments were performed on an Intel Quadcore 2,80 GHz with 4 GB RAM. Instances with 100 up to 1000 items were generated. The coefficients were randomly and independently chosen in the interval $[1, 10n]$ for the first objective function and in $[-10n, -1]$ for the second and third objective function. In contrast to the presented pseudocode, the implementation indicates if a solution is nonextreme supported. The experiments indicate that no nonextreme supported solutions occur, which is most likely for randomly generated instances with a sufficiently large interval for defining the profit values. Hence, the experiments can be used to test Algorithm 3.2.

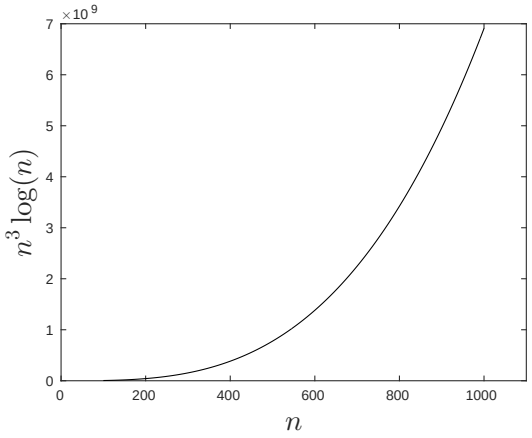
Table 3.4 presents average solution times (t) and numbers of extreme supported solutions ($|\mathcal{X}_{eE}|$) over 30 instances. Algorithm 3.2 solves ($3o_{pnn}$.0c) efficiently. It computes a large number of solutions in a short time. In Figure 3.12, we compare our computational results with the theoretical worst case complexities, i. e., we compare the solution times with $f(n) = n^3 \log n$ (Figures 3.12(a) and 3.12(b)) and the numbers of extreme supported solutions with $f(n) = n^2$ (Figures 3.12(c) and 3.12(d)). Both comparisons show that the experiments fit the theoretical results.

n	t	$\lvert\mathcal{X}_{eE}\rvert$
100	0.05	1 670.43
200	0.74	6 933.00
300	3.87	15 117.60
400	12.78	27 054.57
500	31.16	41 475.03
750	177.39	94 228.83
1000	686.98	168 324.83

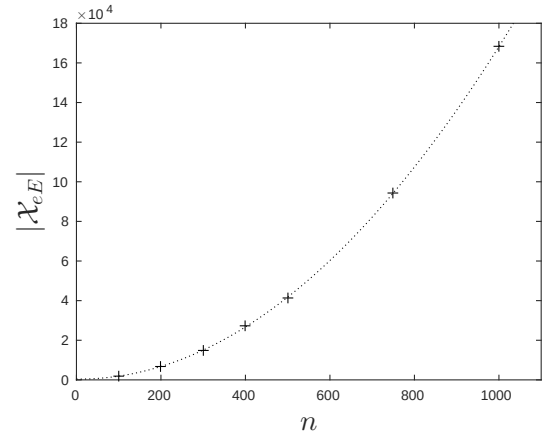
Table 3.4: CPU-times and number of extreme supported solutions for instances of (3o$_{pnn}$.0c) with 100 up to 1000 items (each averaged over 30 instances).



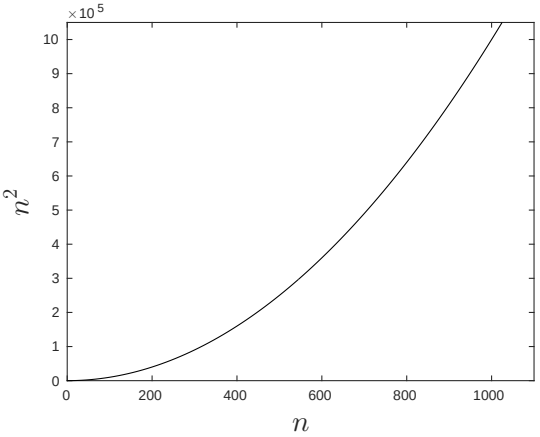
(a) Solution time t depending on number of items n.



(b) Worst case complexity of Algorithm 3.2 is in $\mathcal{O}(n^3 \log n)$.



(c) Number of extreme supported solutions $|\mathcal{X}_{eE}|$ depending on number of items n.



(d) Number of extreme supported solutions for (3o$_{pnn}$.0c) is in $\mathcal{O}(n^2)$.

Figure 3.12: Comparison of experimental and theoretical results for Algorithm 3.2.

3.3.4 The general case: computing extreme supported solutions for multi-objective unconstrained combinatorial optimization problems

The idea of Algorithm 3.2 can be generalized to more complex cases:

In the case of arbitrary weights the hyperplanes do not necessarily intersect the λ_3-axis in the interval $(0, 1)$. They can enter the projected weight space at any of the three bounding edges, or they may not intersect $\mathcal{W}^0$ at all. This is not a problem, since the algorithm by Bentley and Ottmann [1979] is designed for computing the intersections of line segments in general. Therefore, the first step is to compute the line segments $h_i \cap \mathcal{W}^0$, for $i = 1, \ldots, n$. The second difference to Algorithm 3.2 is that without the special structure of (3o$_{pnn}$.0c), the half-spaces h_i^- and h_i^+ are not definitely assigned above and below the hyperplane h_i, respectively, but may also be assigned reverse. Hence, for each intersection point and each item, the sign of the weighted utility has to be computed to define the position vector of the vertex. Overall, the worst case complexity of the algorithm is $\mathcal{O}(n^2 \log n)$, which is the same as for (3o$_{pnn}$.0c).

In the case of higher dimensions with simple arrangements the algorithm of Bentley and Ottmann [1979] cannot be applied since it is limited to the computation of intersection points in the plane. For (mo.0c), with $m > 3$, an alternative is to apply the naïve approach of computing all $\binom{n}{m-1}$ intersection points of $(m-1)$-tuples of hyperplanes and testing whether they are part of the projected weight space or not. To do so, m additional hyperplanes can be introduced that define the boundary of the projected weight space. Only position vectors with correct signs in the corresponding entries are considered. This approach implies an easy strategy for hyperplanes that do not intersect with $(m-2)$ other hyperplanes in the projected weight space (cf. special case of (3o$_{pnn}$.0c): hyperplanes that do not intersect with any other hyperplane in $\mathcal{W}^0$): For $1 \leq t \leq m-1$, all t-faces of the arrangement intersect with $(m-t)$-faces defined by hyperplanes of the boundary of the projected weight space. These intersection points provide the required information. For fixed m, the modifications change the worst case complexity of the algorithm to $\mathcal{O}(n^m)$.

In the case of higher dimensions with non-simple arrangements we distinguish two cases:
(1) If the complete set of extreme and nonextreme supported solutions should be computed, the algorithm can be applied as described above, but the complexity increases to $\mathcal{O}(2^n \cdot n^m)$ due to the possibly exponential number of solutions.
(2) If the nonextreme supported solutions should not be computed, the algorithm has to be modified (as Algorithm 3.2 is a special case of the more general variant belonging to the function in Algorithm 3.3). At first, a filtering of identical hyperplanes should be included. In contrast to (3o$_{pnn}$.0c), the signs of coefficients are not fixed assigned to objective functions. Therefore, the equality of hyperplanes h_i and h_j does not induce equality of the half-spaces h_i^+ and h_j^+. It is also possible, that h_i^+ is equal to h_j^-. This has to be tested

and included in the reconstruction. The second modification handles intersection points λ that have more than $(m-1)$ 0-entries in the position vector, i. e., that are lying on more than $(m-1)$ hyperplanes. Cells adjacent to a k-face and, hence, the extreme supported solutions can be identified if the corresponding position vector has $(m-1-k)$ 0-entries. Therefore, the edges adjacent to λ can be analyzed since the change of the dimension may identify the surplus hyperplane. But depending on the structure, these 1-faces may still have more than $(m-2)$ 0-entries, i. e., that more than $(m-2)$ hyperplanes intersect in one edge. Hence, the algorithm has to recurse one dimension higher to the adjacent 2-faces. Possibly, this has to be continued until the cells are computed explicitly. In the worst case, all k-faces, $0 \leq k \leq m-1$, have to be considered. Including the effort for saving the extreme supported solutions, this can be done in $\mathcal{O}\big((n^{m-1})^{m-1} \cdot n\big) = \mathcal{O}\big(n^{(m-1)^2+1}\big)$, which is still polynomial for fixed m.

3.3.5 Computing all extreme points of a zonotope

The extreme points of a zonotope in $\mathbb{R}^m$ can be computed by examining all cells of the corresponding central arrangement of hyperplanes which is also in $\mathbb{R}^m$. In contrast to the above approach for computing the set of supported solutions, all orthants of $\mathbb{R}^m$ have to be considered as weight space. So, negative weights have to be included and weights equal to zero are also admissible. The associated instance of (mo.0c) can be formulated as described in Section 3.2.1. By doing so, each objective function includes a constant term z^j, for $j = 1, \ldots, m$.

For the computation of the extreme supported solutions, the constant terms are omitted. We examine each orthant separately by adapting the notion of dominance, i. e., by choosing an appropriate combination of maximization and minimization objectives (cf. Section 3.2.3). The above described solution approach for (mo.0c) is applicable for arbitrary weights, therefore, the minimization objectives are transformed into maximization objectives by inverting the sign of all corresponding coefficients. Afterward, the coefficients $p_i^\bullet$ can be used to define the corresponding hyperplanes h_i and the solution approach can be used to compute all extreme supported solutions corresponding to the respective orthant. The extreme points are computed using the original objective functions again including the constant terms.

The centrality of the arrangement of hyperplanes associated to the zonotope allows to concentrate on weights in one half of $\mathbb{R}^m$. Every point y associated to weights with $\lambda_1 > 0$ corresponds to a point $\tilde{y}$ associated to weights with $\lambda_1 < 0$ in the opposite orthant of $\mathbb{R}^m$. The corresponding position vectors are inverse, i. e., $\mathrm{Pos}_i(y) = -\mathrm{Pos}_i(\tilde{y})$, for $i = 1, \ldots, n$, and, hence, also the corresponding extreme supported solutions are inverse, i. e., $x_i = 1 - \tilde{x}_i$, for $i = 1, \ldots, n$. Therefore, the first component λ_1 can be fixed to positive values and the corresponding 2^{m-1} orthants have to be analyzed. To obtain the extreme solutions corresponding to the remaining orthants, every solution has to be inverted.

The solution approach is presented in Algorithm 3.4. The matrix E defines the current combination of maximization and minimization objectives. The solution approach for

(mo.0c) is executed 2^{m-1} times. Hence, for fixed $m \geq 2$, all extreme points of a zonotope can be computed in $\mathcal{O}(n^m)$.

Algorithm 3.4 Algorithm for generating all extreme points of a zonotope in $\mathbb{R}^m$.

Input: profit vectors $p_i^\bullet$, for $i = 1, \ldots, n$, constant vector $z^\bullet \in \mathbb{Z}^m$

$\mathcal{P} := \emptyset$ // *set of extreme points of the zonotope*
$\beta_1 := 1$
for all $(\beta_2, \ldots, \beta_m) \in \{-1, 1\}^{m-1}$ **do** // *for each orthant of* $\mathbb{R}^{m-1}$, $\lambda_1 \geq 0$
    $E := \mathrm{diag}(\beta_1, \ldots, \beta_m)$
    for $i := 1, \ldots, n$ **do**
        generate the hyperplane h_i defined by the profit vector $E \cdot p_i^\bullet$
    compute the set of extreme supported solutions $\mathcal{X}_{eE}$ using $h_1, \ldots, h_n$
    for $x \in \mathcal{X}_{eE}$ **do**
        $\mathcal{P} := \mathcal{P} \cup \left\{\sum_{i=1}^n p_i^\bullet x_i + z^\bullet\right\}$ // *corresponding extreme point*
        for $i := 1, \ldots, n$ **do** // *invert solution*
            $x_i := 1 - x_i$
        $\mathcal{P} := \mathcal{P} \cup \left\{\sum_{i=1}^n p_i^\bullet x_i + z^\bullet\right\}$ // *opposite extreme point for* $\lambda_1 \leq 0$

Output: $\mathcal{P}$

3.4 Efficient computation of extreme supported points of multi-objective knapsack problems

In contrast to (mo.0c), the multi-objective knapsack problem (mo.1c) includes a capacity constraint. Thus, not all subsets of items are feasible in general: the feasible set of (mo.1c) is a subset of that of (mo.0c). As a consequence, the solution approach presented in Section 3.3.4 is not directly applicable. More precisely, since efficient solutions of (mo.0c) may become infeasible for (mo.1c), other feasible solutions may become efficient for (mo.1c).

Nevertheless, the projected weight space still provides all information for computing the set of extreme supported solutions using the concept of weight space decomposition (see Section 3.1.3 and Przybylski et al. [2010a]). The projected weight space can be decomposed into convex polytopes such that the weighted sum scalarization using weights that correspond to one polytope have the same optimal solutions. Different from Section 3.3, the polytopes are not generated by an arrangement of hyperplanes. Nevertheless, the definitions of k-faces, cells, facets, edges, vertices and subfaces can be applied analogously.

Note that for (mo.1c) there may exist *equivalent* extreme supported efficient solutions, i. e., extreme supported solutions x and x' that correspond to the same extreme supported nondominated point $f(x) = f(x')$. Therefore, we focus on computing (extreme) supported points and one corresponding solution in the following.

Several approaches for computing the set of extreme supported points for (mo.1c) have been suggested in the literature (see Section 3.4.1). We show how these algorithms can be improved using the results on unconstrained problems in Section 3.4.2. As in Section 3.3.3, we perform a case study on tri-objective problems with one positive and two negative objectives in Section 3.4.3 including computational tests.

3.4.1 Literature review on multi-objective approaches for decomposing the weight space

Benson and Sun [2002] present an algorithm that uses a weight space decomposition to find all extreme supported points of multi-objective linear optimization problems (MOLP). The algorithm makes use of a duality between the objective space and the weight space. In each iteration of the algorithm of Benson and Sun [2002], either a weight vector $(\lambda_1, \ldots, \lambda_m)^\top$ is detected that leads to a new extreme supported point or the algorithm stops since no such weight vector exists. Each computed extreme supported point corresponds to a face of the weight space. Since (MOLP) is linear, a weight vector that is not an element of these polytopes can be computed solving a linear inequality system. This weight vector certainly corresponds to a new extreme supported point of (MOLP). If no such weight vector exists, i. e., if the linear system is infeasible, the algorithm stops. The set of extreme supported points correspond to the complete weight space decomposition, i. e., every weight vector of the weight space corresponds to at least one of the extreme points. All extreme points have been computed.

The idea of the approach of Benson and Sun [2002] can be combined with the concept of dichotomic search (cf. Sections 2.1 and 4.2) for an application on multi-objective integer optimization problems (MOIP). We first present the general structure of such an approach, see Algorithm 3.5, and then give a review on the literature.

A general approach For (MOIP), another concept for selecting new weights has to be developed. Problems with three or more objective functions, computing only nondominated extreme points causes some challenges that we explain later on. But therefore, we first show how to compute all extreme points of the convex hull $\text{conv}(\mathcal{Y})$ of feasible points. To do so, we extend the weight space to $\mathbb{R}^m$. In the following, we refer to faces of $\text{conv}(\mathcal{Y})$ as *primal faces* and to faces of the weight space decomposition as *dual faces*. Every vector that is normal to a primal face corresponds to a weight vector in the weight space $\mathbb{R}^m$. As in the dichotomic search approach, the weights defined by these vectors are used to find new extreme points or exclude primal faces from further consideration.

We assume that an initial set of $(m+1)$ affinely independent extreme points is available as an input for Algorithm 3.5. A naïve approach to obtain this initial set is to solve several weighted sum problems using arbitrary weights until $(m+1)$ affinely independent extreme points have been found. Using this set of extreme points, an initial convex hull can be computed. The vectors normal to the primal facets define the weights for applying the weighted sum scalarization. One weighted sum problem is solved in each iteration and,

Algorithm 3.5 Algorithm for generating all extreme points of the convex hull conv($\mathcal{Y}$) of feasible points for (MOIP).

Input: initial set Y of $(m+1)$ affinely independent extreme supported points
1: compute convex hull conv(Y)
2: define $\mathcal{F}$ as the set of primal facets and include all facets of conv(Y) in $\mathcal{F}$
3: **for** $F \in \mathcal{F}$ **do**
4: let λ be the vector normal to F
5: define (P) as the weighted sum scalarization of MOIP with weight vector λ
6: compute an optimal point y of (P)
7: **if** $y \notin Y$ **then**
8: $Y := Y \cup \{y\}$
9: update convex hull conv(Y)
10: include new facets of conv(Y) in $\mathcal{F}$
11: delete F from $\mathcal{F}$
Output: Y, set of extreme points of conv($\mathcal{Y}$) and, possibly, some nonextreme points on the boundary of conv($\mathcal{Y}$)

either, a new point is found and new primal facets enter the process (Steps 8 and 10 of Algorithm 3.5), or the extreme point was already known. In both cases this primal facet can be excluded from consideration. The algorithm stops if all primal facets have been analyzed. If nonextreme points on the boundary of conv($\mathcal{Y}$) exist, for example nonextreme supported points, some of these may be found by the corresponding weighted sum scalarization during the process. This does not affect the algorithm, but generates additional information. For generating the convex hull of a given set of points in $\mathbb{R}^m$, for example, the quickhull algorithm of Barber et al. [1996] can be applied. For a recent survey on convex hull algorithms see, for example, de Berg et al. [2008]. Algorithm 3.5 summarizes the approach.

A generalization of the dichotomic search for computing all extreme supported points (i. e., now we concentrate on nondominated points) of (mo.1c), for $m > 2$, is not easy. Przybylski et al. [2010a] discuss in detail possible shortcomings of Algorithm 3.5, including problems that may occur during the initialization as well as the optimization for a given hyperplane in the objective space. One problem is that it is unclear how to compute initial points or how to define an initial primal facet. Furthermore, the weight vector corresponding to a primal facet may have negative entries. In this case, on the one hand, it cannot be guaranteed to find extreme supported points using this weight vector. But, on the other hand, by excluding these weight vectors it cannot be guaranteed to find all extreme supported points: For $m > 2$, an extreme supported point may not be adjacent to nondominated facets of the convex hull conv($\mathcal{Y}$) but only to nondominated k-faces with $k < m-1$. So it may not be found using only weight vectors corresponding to facets and having only positive entries.

Przybylski et al. [2010a] reinterpret the dichotomic search and concentrate on the decomposition of the normalized weight space for computing all extreme supported points of multi-objective integer optimization problems. By computing the common subfaces of pairs of cells, they represent the complete normalized weight space and guarantee to find all extreme supported points. For $m = 3$, the subfaces are edges which can be investigated by solving corresponding bi-objective problems. For higher dimensions, i. e., $m \geq 4$, the approach is applied recursively.

Özpeynirci and Köksalan [2010] develop an algorithm for finding all extreme supported points of multi-objective mixed integer optimization problems on the basis of the ideas of Aneja and Nair [1979]. They assume that the problem has m minimization objectives and that all components of all feasible points are positive, i. e., that $f_j(x) > 0$ for all $j = 1, \ldots, m$ and for all $x \in \mathcal{X}$. The algorithm operates on the nondominated frontier, i. e., the union of all nondominated faces of $\text{conv}(\mathcal{Y})$, in the objective space. To overcome the difficulties of a multi-objective dichotomic search described above, Özpeynirci and Köksalan [2010] introduce dummy points in the objective space:

$$d_j = De_j, \qquad j = 1, \ldots, m \tag{3.1}$$

with e_j the j-th unit vector of $\mathbb{R}^m$, and $D \in \mathbb{R}_>$ a sufficiently large positive constant in the sense that all extreme supported points remain extreme supported for the extended set $\text{conv}(\mathcal{Y} \cup \{d_1, \ldots, d_m\})$. It can be shown, that such a constant D always exists. The inclusion of the dummy points slightly changes the shape of the nondominated frontier in the objective space. In general, the nondominated frontier consists of a union of nondominated faces of different dimensions. Including the dummy points allows to define the nondominated frontier as a union of nondominated facets. Accordingly, the corresponding weight space decomposition is modified. Özpeynirci and Köksalan [2010] verify that for weight values very close to the boundary of the weight space, i. e., with at least one component smaller or equal to a predefined value $\epsilon > 0$, the corresponding weighted sum problem is solved by a dummy point. Hence, the polytopes in the weight space corresponding to the original extreme supported points do not intersect the boundary of the weight space.

The algorithm of Özpeynirci and Köksalan [2010] initializes the search with the m dummy points and the corresponding facet and weight vector. The weighted sum problem is solved and either the resulting point was already known, then the corresponding facet is part of the nondominated frontier and can be stored, or a new extreme supported point has been computed and the list of facets has to be updated. The algorithm builds the new facets by combining the newly generated point with all subsets of points with cardinality $(m-1)$ of the current facet. By doing so, in three or higher dimensions negative weight vectors may occur. In this case, Özpeynirci and Köksalan [2010] propose to select one of the already known extreme supported points that do not define the facet that corresponds to the current weight vector and to generate m new facets using this point. An alternative would be to use the quickhull algorithm of Barber et al. [1996] to update the convex hull, which would only produce positive weights due to the dummy points. However, the

algorithm investigates each facet at most once and, hence, terminates after a finite number of iterations.

Bökler and Mutzel [2015] present an approach for enumerating all extreme supported points of multi-objective combinatorial optimization problems, which is based on dual variants of Benson's algorithm for multi-objective linear optimization (see Ehrgott et al. [2012] and Heyde and Löhne [2008]). We adapt the notation of Bökler and Mutzel [2015] to the notation for (mo.1c). A primal polyhedron is given by $\mathcal{P} = \mathcal{Y} - \mathbb{R}^m_{\geq}$. The extreme supported points of (mo.1c) are the extreme points of $\mathcal{P}$. A dual polyhedron $\mathcal{D}$ is defined in $\mathbb{R}^m$, where the first $(m-1)$ coordinates represent the projected weight space $\mathcal{W}^0$ ($(\mu_1, \dots, \mu_{m-1})^\top := (\lambda_2, \dots, \lambda_m)^\top$) and the m-th coordinate μ_m is associated to the objective value of the corresponding weighted sum scalarization of (mo.1c). Every extreme point y of $\mathcal{P}$ defines a hyperplane

$$h(y) = \left\{ \mu \in \mathbb{R}^m \,:\, \mu_m = \left(1 - \sum_{j=2}^{m} \mu_{j-1}\right) y_1 + \sum_{j=2}^{m} \mu_{j-1} y_j \right\}.$$

The dual polyhedron $\mathcal{D}$ consists of all points in $\mathbb{R}^m$ that are located above all of these hyperplanes. More precisely, for each extreme point y of $\mathcal{P}$ a half-space

$$H(y) = \left\{ \mu \in \mathbb{R}^m \,:\, \mu_m \geq \bar{\mu}_m \text{ for } \bar{\mu} \in h(y) \text{ with } \bar{\mu}_j = \mu_j, \forall j \in \{1, \dots, m-1\} \right\}$$

is defined and a weight set

$$\mathcal{W} = \left\{ \mu \in \mathbb{R}^m \,:\, \mu_j \geq 0, \forall j \in \{1, \dots, m-1\}, 1 - \sum_{j=2}^{m} \mu_{j-1} \geq 0 \right\}.$$

The dual polyhedron $\mathcal{D}$ corresponds to the intersection of all half-spaces $H(y)$ and $\mathcal{W}$.

The duality of the two polyhedra $\mathcal{P}$ and $\mathcal{D}$ can easily be derived using the linear programming relaxation of the weighted sum scalarization of (mo.1c). Each facet of the lower bound of $\mathcal{D}$ is part of one defining hyperplane $h(y)$ and, hence, corresponds to one extreme point of $\mathcal{P}$. The algorithm of Bökler and Mutzel [2015] works as follows: Starting with a polyhedron that contains $\mathcal{D}$ at every extreme point of this polyhedron a ray is shot into the dual polyhedron, i. e., a weighted sum scalarization of the optimization problem is solved for the respective weight values, with optimal solution x. If this solution corresponds to the same extreme point of $\mathcal{D}$ again, then this point is also extreme for the final polyhedron $\mathcal{D}$ and can be excluded from further consideration. Otherwise, the newly generated solution x is at least weakly efficient and the corresponding point y is saved as candidate for a extreme supported point in $\mathcal{P}$. Then, the corresponding hyperplane and half-space is constructed and intersected with the current dual polyhedron $\mathcal{D}$. Newly generated extreme points of $\mathcal{D}$ are included in the process. Finally, if all extreme points of $\mathcal{D}$ have been analyzed, the final dual polyhedron $\mathcal{D}$ is obtained and, hence, all extreme supported points of $\mathcal{P}$ are known. In a last step, all weakly nondominated but nonextreme

points have to be filtered out. As an alternative, Bökler and Mutzel [2015] also present a lexicographic version to avoid weakly nondominated points that may occur in the case of weight vectors with entries equal to 0.

Przybylski et al. [2017] introduce a straightforward dichotomic search algorithm and two different variants for computing extreme supported points of multi-objective integer linear programming problems. The first algorithm essentially corresponds to Algorithm 3.5, allowing negative weights and resulting in the computation of all extreme points of the convex hull of the set of feasible points $\text{conv}(\mathcal{Y})$. As a final step, all dominated extreme points have to be filtered out. Przybylski et al. [2017] introduce different initializations and concepts for reducing the number of iterations in two alternative algorithms.

The first variant makes use of the concept of dummy points similar to Özpeynirci and Köksalan [2010]. The algorithm is initialized by a set composed of the m dummy points and one extreme supported point computed for arbitrary positive weights, e. g., for equally weighted objective functions. The associated convex hull has solely facets with positive coefficients in the normal vectors, except for the one facet defined by the m dummy points. This facet is excluded from the further process. Starting with the remaining m facets, the following steps can be performed as in the basic algorithm. Przybylski et al. [2017] show that this variant generates only supported points and that all extreme supported points are generated by the algorithm. A disadvantage of the approach is, that in practice the large value required for D in the definition of the dummy points (see Equation (3.1)) can lead to numerical imprecision.

The second variant generates an initial set of nondominated points in yet another way. Subproblems of (mo.1c) with less objective functions, i. e., where one to m objective functions have been deleted, are defined. Przybylski et al. [2010b] show that an extreme supported solution of the subproblem is already extreme supported for the original problem if it is efficient for the original problem. Thus, all m subproblems with $(m-1)$ objective functions are solved and the union of all computed extreme supported points is used to initialize the algorithm. Similar to the dummy points of the previous approach, these points are optimal for weighted sum problems with weights on the boundary of the weight space. The algorithm proceeds as before by computing the convex hull of the initial set of points. Only facets that define positive weights are evaluated. Przybylski et al. [2017] showed, that this guarantees that only supported points are generated and that all extreme supported points will be identified by the algorithm. Some initial points may be dominated in the original problem with m objectives. Hence, these points should be filtered out at the end.

3.4.2 Multi-objective knapsack problems

As described above, the results of Section 3.3 are not directly transferable to the constrained problem (mo.1c). However, the nondominated frontier of the unconstrained

problem provides an upper bound set on the nondominated frontier of the same problem with a capacity constraint (see Figure 3.13(a)). Efficient solutions of (mo.0c) that do not violate the capacity constraint are also efficient for (mo.1c). Thereby, a cell of the arrangement of hyperplanes associated to those solutions is contained in the polytope of the weight space decomposition associated to (mo.1c) for the same solution. Hence, the arrangement of hyperplanes associated to (mo.0c) can be computed in a preprocessing step and all resulting feasible points can be used to initialize a multidimensional dichotomic search. All faces of the arrangement corresponding to infeasible solutions have to be analyzed again. We call these faces *infeasible faces*. Similarly, connected parts of infeasible faces in the projected weight space are called *infeasible areas*. Inversely, all faces corresponding to feasible solutions are called *feasible faces* of the arrangement associated to (mo.1c).

For simple arrangements, the position vectors of cells $\varphi_1^{(m-1)}$ and $\varphi_2^{(m-1)}$ with a common subface $\varphi^{(m-2)}$ differ in exactly one position i that corresponds to the hyperplane h_i which separates the cells. We assume that $\varphi_1^{(m-1)}$ is a feasible cell and that $\varphi_2^{(m-1)}$ is an infeasible cell for (mo.1c). Hence, the common subface $\varphi^{(m-2)}$ corresponds to one feasible and one infeasible solution. The two solutions also correspond to the subfaces of $\varphi^{(m-2)}$ and so on. We call k-faces $\varphi^{(k)}$, $0 \leq k \leq m-2$, *critical faces* if they correspond to at least one feasible and at least one infeasible solution for (mo.1c). For non-simple arrangements, more solutions may correspond to a k-face than in the simple case, but still a face is called critical if feasible and infeasible solutions are among them. Trivially, feasible and infeasible areas are separated by critical faces.

Example 3.21

(a) Consider the following bi-objective knapsack problem:

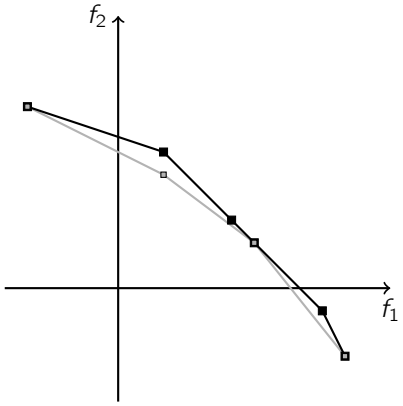
$$\begin{array}{lll} \max & -x_2 + 3x_3 - 4x_4 + 6x_5 - 5x_6 + x_7 \\ \max & x_1 + 2x_2 - 3x_3 + 4x_4 - 2x_5 - x_6 + x_7 \\ \text{s.t.} & 5x_1 + 3x_2 + 2x_3 + 4x_4 + 5x_5 + 1x_6 + 4x_7 \leq 17 \\ & x_i \in \{0,1\}, \qquad i = 1,\ldots,7. \end{array}$$

This is a constrained version of the bi-objective unconstrained combinatorial optimization problem of Example 3.14. In Figure 3.13(a), the respective nondominated frontiers are shown. Each point in the nondominated frontier of the constrained problem is dominated by a point in the nondominated frontier of the unconstrained problem.

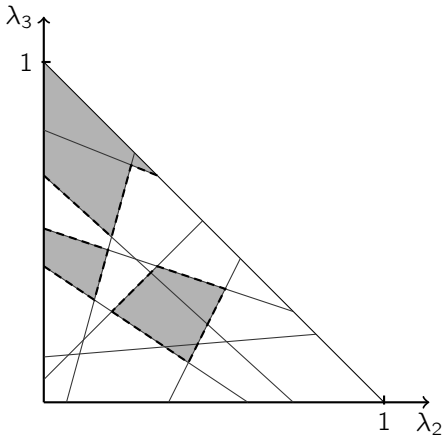
(b) Consider the following tri-objective knapsack problem:

$$\begin{array}{lll} \max & -72x_1 - 22x_2 + 46x_3 - 36x_4 - 8x_5 + 4x_6 + 11x_7 + 22x_8 \\ \max & -42x_1 + 8x_2 + 16x_3 + 24x_4 - 13x_5 + 64x_6 - 154x_7 - 38x_8 \\ \max & 18x_1 + 11x_2 - 44x_3 + 54x_4 + 52x_5 - 56x_6 + 56x_7 + 52x_8 \\ \text{s.t.} & 13x_1 + 6x_2 + 8x_3 + 8x_4 + x_5 + 2x_6 + 13x_7 + 10x_8 \leq 26 \\ & x_i \in \{0,1\}, \qquad i = 1,\ldots,8. \end{array}$$

The unconstrained version of this instance has 25 extreme supported solutions. Seven of the cells of the associated arrangement of hyperplanes become infeasible as soon as the constraint is included, see Figure 3.13(b).



(a) Supported points and nondominated frontiers of the constrained (gray) and unconstrained (black) problem.



(b) Arrangement of hyperplanes associated to the unconstrained version of the instance. Including the constraint, all faces inside the gray shaded areas correspond to infeasible solutions. The dashed line segments highlight the critical edges and vertices.

Figure 3.13: Objective space (a) and projected weight space (b) corresponding to Examples 3.21(a) and (b), respectively.

In Algorithm 3.6 an approach for computing all extreme supported points of (mo.1c) is presented, using the concepts of Section 3.3.4 for unconstrained problems as a preprocessing. All of the multi-objective approaches for decomposing the weight space presented in Section 3.4.1 can be used in Steps 3 and 4 to compute all extreme supported points of (mo.1c). In the following, we discuss the implementation of Algorithm 3.6 in combination with these approaches.

Algorithm 3.6 Generates all extreme supported points of (mo.1c)

Input: (mo.1c)

1: compute the set of extreme supported solutions $\mathcal{X}_{eE}$ of the unconstrained version of (mo.1c) using concepts of Section 3.3.4

2: $\mathcal{Y}_{eN} := \{y \in \mathbb{Z}^m : y = f(x), x \in \mathcal{X}_{eE} \text{ with } \sum_{i=1}^{n} w_i x_i \leq W\}$ // *filter feasible points*

3: choose any algorithm of Section 3.4.1 and use all points $y \in \mathcal{Y}_{eN}$ to initialize it

4: apply the chosen algorithm to complete $\mathcal{Y}_{eN}$

Output: $\mathcal{Y}_{eN}$

Przybylski et al. [2010a] Utilizing the information, precomputed in Steps 1 and 2 of Algorithm 3.6 in combination with the algorithm of Przybylski et al. [2010a] is quite natural. All feasible areas in the projected weight space can be considered as explored and the algorithm can start with examining the critical faces.

Özpeynirci and Köksalan [2010] The method of Özpeynirci and Köksalan [2010] first has to be adapted to the problem structure of (mo.1c), i. e., the definition of the dummy points has to be reconsidered. Two facts have to be kept in mind:

- (mo.1c) is a maximization problem.
- Components of feasible points may be negative, and hence the origin can in general not be selected as reference point. Instead, an utopian point $d^0 = (d_1^0, \dots, d_m^0)^\top$, with $f_j(x) < \sum_{i=1}^n \max\{p_i^j, 0\} = d_j^0$ for all $x \in \mathcal{X}$ and for $j = 1, \dots, m$, can be used.

Thus, for (mo.1c) the dummy points of Özpeynirci and Köksalan [2010] can be defined as

$$d_j = d^0 - De_j, \qquad j = 1, \dots, m$$

with $D \in \mathbb{R}_>$ a sufficiently large positive constant. The existence of D can be shown analogously to the existence of D for Equation 3.1.

Then, in Step 3 of Algorithm 3.6, the algorithm of Özpeynirci and Köksalan [2010] can be initialized using the m dummy points and, additionally, using all feasible points found in Step 1. I. e., the convex hull of all of these points has to be built. All facets in the objective space that correspond to feasible vertices in the projected weight space are part of the nondominated frontier and can be stored. All other facets are used to start the algorithm in Step 4.

Bökler and Mutzel [2015] The initial polyhedron in the algorithm of Bökler and Mutzel [2015] can be reduced by intersecting with all half-spaces $H(y)$ corresponding to the precomputed extreme supported points y of Steps 1 and 2. This polyhedron contains $\mathcal{D}$ and the algorithm can be applied as described in Step 4. Furthermore, all extreme points of the polyhedron that correspond to feasible vertices in the projected weight space are already extreme points of $\mathcal{D}$ and do not need to be analyzed.

Przybylski et al. [2017] Both enhanced solution approaches of Przybylski et al. [2017] can easily be applied in this context by including the precomputed feasible points in the initial convex hull.

Obviously, the efficiency of the preprocessing is directly related to the slackness in the capacity constraint. If the constraint is tight, many solutions of (mo.0c) may become infeasible and it is not beneficial to compute them. Otherwise, the initial information is very helpful. Another criterion is the distribution of positive and negative coefficients in the objective functions. Usually, all coefficients are assumed to be positive for knapsack

problems. In this case, the unconstrained problem has one single optimal solution for all possible weight values for the weighted sum scalarization, namely to include every item. Due to the general assumption that the capacity W is smaller then the sum of all weights w_i, $i = 1, \dots, n$, this solution is infeasible for the knapsack problem and the preprocessing does not offer any useful information. In Section 3.4.3, we present computational experiments that show the efficiency of this preprocessing for problems of a special structure.

3.4.3 Case study: tri-objective knapsack problems with one positive and two negative objective functions

In this section, we again demonstrate the above results on a more specific problem with three objectives. We assume that the first objective has positive coefficients and the second and third objective have negative coefficients.

$$\begin{aligned} \text{vmax} \quad & f(x) = \left(\sum_{i=1}^{n} p_i^1 x_i \,,\, \sum_{i=1}^{n} p_i^2 x_i \,,\, \sum_{i=1}^{n} p_i^3 x_i \right) \\ \text{s.t.} \quad & \sum_{i=1}^{n} w_i x_i \leq W \\ & x_i \in \{0,1\}, \qquad i = 1, \dots, n \end{aligned} \tag{3o$_{pnn}$.1c}$$

with $p_i^1 > 0$, $p_i^2 < 0$, $p_i^3 < 0$, and $w_i > 0$ for all $i = 1, \dots, n$.

As discussed in Section 3.3.3, the unconstrained version of (3o$_{pnn}$.1c), the problem (3o$_{pnn}$.0c), has a specially structured weight space decomposition. For simplicity, we again use the terms *above* and *below* to describe the relations of cells to hyperplanes and intersection points as defined in Section 3.3.3. In the upper left part of the weight space where the first (positive) objective function is only marginally weighted, the supported solution x corresponding to the cell $\varphi_0^{(2)}$ is $x = \mathbf{0}_n$. I. e., none of the items is included, which is certainly feasible for (3o$_{pnn}$.1c). If we assume that there do not exist identical hyperplanes, which we do in the following, we can conclude that solutions corresponding to cells that are adjacent to the same edge differ in exactly one item. Due to the fact that all half-spaces h_i^+ are lying below the corresponding hyperplanes, all cells above the separating hyperplane correspond to solutions with $x_i = 0$ and all cells below the hyperplane correspond to solutions with $x_i = 1$. Since every hyperplane has a negative slope, it is possible to define a path of cells from $\varphi_0^{(2)}$ to any cell, where at each step, from one cell to the next, one item is included. Note that this path is, in general, not unique. Trivially, if a cell is feasible, then all previous cells on this path are also feasible. Accordingly, if a cell is infeasible, then all subsequent cells on a path are also infeasible.

The cell $\varphi_1^{(2)}$ which has $(\lambda_2, \lambda_3)^\top = (0,0)^\top$ in its boundary corresponds to the solution $x = \mathbf{1}_n$ since the second and third (negative) objective function are only marginally weighted. This solution is certainly infeasible for (3o$_{pnn}$.1c) (unless $\sum_{i=1}^{n} w_i \leq W$). Analogously to the above discussion, starting with $\varphi_1^{(2)}$, a path to every other cell of the

arrangement can be defined where one variable is set to 0 at each step. If one cell is infeasible, then every other cell on this path up to the cell in question is also infeasible. If one cell is feasible, then all subsequent cells on the path are also feasible.

Concluding, all feasible cells are connected as well as all infeasible cells are connected. The weight space consists of one feasible and one infeasible area. A set of critical faces separates the two areas and the set of critical faces is also connected.

Example 3.22 *The following instance of* (3o$_{pnn}$.1c) *is a constrained version of Example 3.19:*

$$\begin{aligned}
\max \quad & 16x_1 + 21x_2 + 10x_3 + 9x_4 + 3x_5 \\
\max \quad & -24x_1 - 14x_2 - 10x_3 - x_4 - 27x_5 \\
\max \quad & -4x_1 - 9x_2 - 10x_3 - 21x_4 - 12x_5 \\
\text{s.t.} \quad & 4x_1 + 3x_2 + 3x_3 + 2x_4 + 1x_5 \leq 5 \\
& x_i \in \{0,1\}, \qquad i = 1,\dots,5.
\end{aligned}$$

The constraint excludes five of the extreme supported solutions of Example 3.19. Figure 3.14 shows the subdivision of the projected weight space in one feasible and one infeasible area. The two cells corresponding to the solutions $(1,1,0,0,0)^\top$ *with a total weight of* 7 *and* $(0,1,1,1,0)^\top$ *with a weight of* 8 *mark the beginning of the infeasible part for any path from* $\varphi_0^{(2)}$ *to* $\varphi_1^{(2)}$*. One such path, subsequently including items* 2, 4, 3, 1, *and* 5*, is illustrated in the figure.*

The algorithm Due to the structure of (3o$_{pnn}$.1c), Steps 1 and 2 of Algorithm 3.6 can be combined by concentrating on the feasible area of the projected weight space. Algorithm 3.2 (see Section 3.3.3) can be adapted to solely analyze the feasible area of the arrangement of hyperplanes corresponding to the unconstrained version of (3o$_{pnn}$.0c) The idea is to take advantage of the critical vertices that are computed during the process. The set of hyperplanes is divided into three subsets that have to be updated iteratively. In the following, we use the terms feasible, critical and infeasible hyperplanes. This is, of course, no appropriate classification, since a hyperplane may intersect the feasible or infeasible area several times and, hence, may also overlap with several critical faces. What we mean is that, for a given value of λ_2, the point (λ_2, λ_3) on this hyperplane is part of a feasible, infeasible or critical face. Thus, during the process, the hyperplanes are classified as feasible, infeasible or critical, which simplifies the following description. For a fixed value of λ_2, one or more hyperplanes can be classified as critical: If λ_2 corresponds to a critical vertex, then all hyperplanes that intersect at this point are classified as critical. Otherwise, λ_2 corresponds to a point on a critical edge, which is assigned to exactly one hyperplane.

In Algorithm 3.2, a permutation $\pi \in \mathcal{S}_n$ is defined that represents an order relation of all hyperplanes. The permutation is used to compute intersection points of hyperplanes systematically. Recall that the algorithm treats the intersection points from the left to the right, i. e., with increasing values of λ_2. The permutation π is only kept updated for the feasible hyperplanes and, in addition, the critical hyperplane. If the analyzed intersection

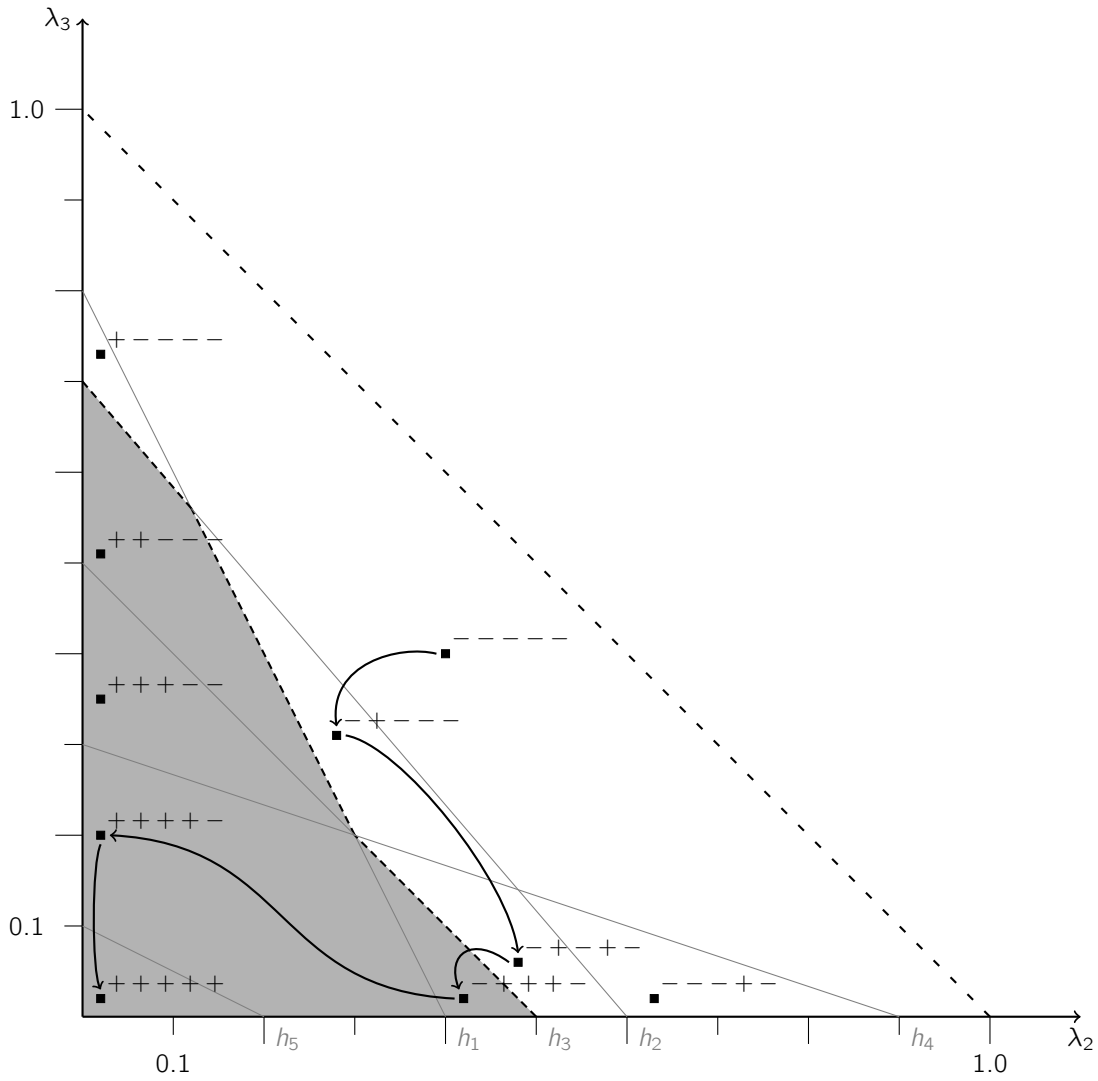


Figure 3.14: Projected weight space for the unconstrained version of Example 3.22, separated into feasible (white) and infeasible (gray) area. The critical edges are dashed. A path from $\varphi_0^{(2)}$ to $\varphi_1^{(2)}$ is displayed. On this path, the decision of including item 3 makes the solution infeasible.

point is critical, i. e., corresponds to two or more critical hyperplanes, the algorithm tests which hyperplane corresponds to the critical edge that is adjacent to this vertex and has larger values of λ_2. It marks the position c of the item $\pi(c)$ as critical. All feasible hyperplanes have positions $j < c$. The infeasible hyperplanes are omitted, i. e., the permutation is not updated for elements $\pi(j)$, with $c < j \leq n$. But, as described above, the sets are not permanent and the position of the critical hyperplane has to be updated iteratively. An infeasible hyperplane can only become critical or feasible by intersecting the critical hyperplane and, the other way round, a feasible hyperplane can only become critical or infeasible by intersecting the critical hyperplane. As a consequence, three amendments of Algorithm 3.2 are necessary:

1. *Initialization:* Besides the set of hyperplanes h_i, for $i = 1, \ldots, n$, the weight coefficients w_i, for $i = 1, \ldots, n$, and the capacity W are required as input data. The initialization of Algorithm 3.2 (Lines 1 to 10) can be stopped as soon as the newly generated solution is infeasible. The corresponding hyperplane builds the set of critical hyperplanes, all hyperplanes that have already been treated are feasible and the remaining ones are infeasible. Finally, an intersection point of the critical hyperplane has to be computed according to the algorithm of Bentley and Ottmann [1979]. We explain this in the context of Amendment 3 below when discussing the *computation of intersection points*.

2. *Generation of solutions and update of the permutation:* The intersection points, which are vertices of the arrangement of hyperplanes, can be feasible, critical or infeasible. The evaluation has to be adapted to the respective cases.

   (a) If the intersection point is feasible, i. e., if both intersecting hyperplanes are feasible, the procedures GenerateSolutions and UpdatePermutation of Algorithm 3.2 can be applied as before.

   (b) If the intersection point is critical, i. e., if the critical hyperplane is involved, every generated solution has to be checked for feasibility. This test shows whether the sets of critical, feasible, and infeasible hyperplanes have to be updated. Let us assume that the critical hyperplane $h_{\pi(j)}$ and one feasible hyperplane $h_{\pi(j-1)}$ or one infeasible hyperplane $h_{\pi(j+1)}$ intersect, i. e., including all items corresponding to feasible hyperplanes and not including all other items build a supported solution and adding item $\pi(j)$ is not feasible, as well as adding item $\pi(j+1)$ afterward. As shown in Figure 3.15, there exist four possibilities:
   I. Intersection of $h_{\pi(j)}$ and $h_{\pi(j-1)}$, where $h_{\pi(j)}$ remains critical, i. e., after excluding item $\pi(j-1)$, it is still not feasible to include item $\pi(j)$. Then hyperplane $h_{\pi(j-1)}$ switches from the feasible to the infeasible set.
   II. Intersection of $h_{\pi(j)}$ and $h_{\pi(j-1)}$, where $h_{\pi(j)}$ turns from critical to feasible, i. e., after excluding item $\pi(j-1)$, item $\pi(j)$ can be included without violating the capacity constraint. Then hyperplane $h_{\pi(j-1)}$ switches from the feasible to the critical set.

III. Intersection of $h_{\pi(j)}$ and $h_{\pi(j+1)}$, where $h_{\pi(j)}$ remains critical, i. e., after excluding item $\pi(j)$, it is feasible to include item $\pi(j+1)$. Then, hyperplane $h_{\pi(j+1)}$ switches from the infeasible to the feasible set.
IV. Intersection of $h_{\pi(j)}$ and $h_{\pi(j+1)}$, where $h_{\pi(j)}$ turns from critical to infeasible, i. e., after excluding item $\pi(j)$, it is still not feasible to include item $\pi(j+1)$. Then hyperplane $h_{\pi(j+1)}$ switches from the feasible to the critical set.
As a consequence, the update of the permutation (procedure UpdatePermutation of Algorithm 3.2) has to be adapted to these changes. If more than two hyperplanes intersect in a critical vertex, more cases have to be taken into consideration.

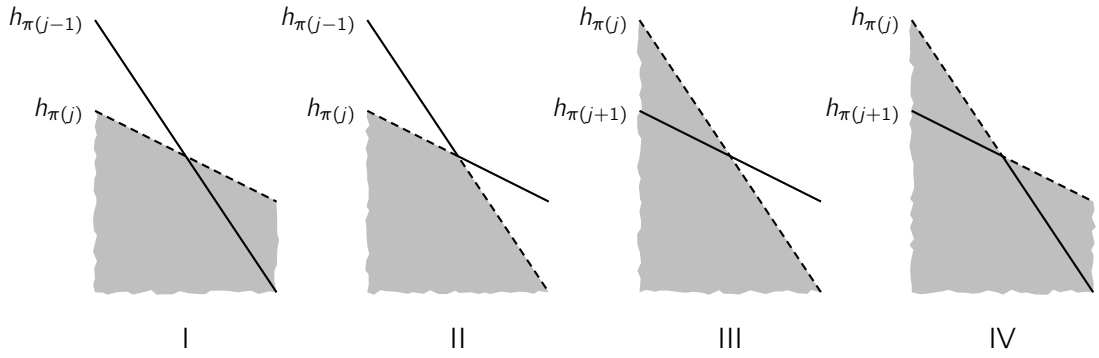


Figure 3.15: Four possible changes (I to IV) of the sets of critical, feasible and infeasible hyperplanes at an intersection point of hyperplanes, where the current critical hyperplane (dashed) is involved.

(c) If the intersection point is infeasible, i. e., if both intersecting hyperplanes are infeasible, then the evaluation can be skipped. Neither does a feasible solution correspond to an adjacent cell, nor has the permutation to be updated.

3. *Computation of intersection points:* Also for the update of the list of intersection points (procedure ComputeIntersection of Algorithm 3.2) we have to distinguish different cases. Recall that the new intersection points are computed after the update of the feasible, critical and infeasible set and the permutation π.

   (a) The considered hyperplane is feasible, i. e., the current intersection point is already feasible or the hyperplane is the feasible one of the cases II or III. As in Algorithm 3.2, the intersection point of the new predecessor or successor in the permutation has to be computed.

   (b) The considered hyperplane is critical, i. e., it is the critical hyperplane of the cases I to IV. A critical hyperplane needs a special treatment since infeasible hyperplanes can become critical or feasible by intersecting it. Thus, all intersections of the critical hyperplane and infeasible hyperplanes have to be computed. As before, only intersection points that have a value of λ_2 that is larger than

that of the current intersection point are relevant. Among those, the intersection point that has the smallest value of λ_2 is saved in the List Λ of intersection points. This corresponds to case of computing the intersection of the hyperplane $\pi(j+t)$ and the hyperplane below it (Line 18 of Algorithm 3.2), though here the hyperplane below has to be identified among the whole set of infeasible hyperplanes.

In cases I and IV, the critical hyperplane has also a new predecessor and the corresponding intersection point has to be computed. Depending on the structure of the arrangement of hyperplanes, one or both intersection points may be relevant, i. e., feasible or critical. We present three examples: The critical hyperplane $h_{\pi(j)}$ in Case V of Figure 3.16 first intersects with one of its successors in π, i. e., with an infeasible hyperplane, and $h_{\pi(j)}$ becomes infeasible. The intersection of $h_{\pi(j)}$ and its predecessor $h_{\pi(j-1)}$ also has been computed, but it is infeasible. As discussed in Amendment 2c, this intersection point will be skipped. The critical hyperplane $h_{\pi(j)}$ in Case VI of Figure 3.16 first intersects with its predecessor $h_{\pi(j-1)}$ in π, and $h_{\pi(j)}$ stays critical. The intersection of $h_{\pi(j)}$ and one of its successors also has been computed and will be analyzed in the further process. This remains true if another feasible hyperplane $h_{\pi(j-1)}$ first intersect $h_{\pi(j)}$ (see, for example, Case VII in Figure 3.16). The hyperplane $h_{\pi(j-1)}$ becomes critical or infeasible, which implies that $h_{\pi(j)}$ becomes feasible or stays critical and, as before, at least one of the computed intersection points is critical or feasible. In the example both intersection points are feasible.

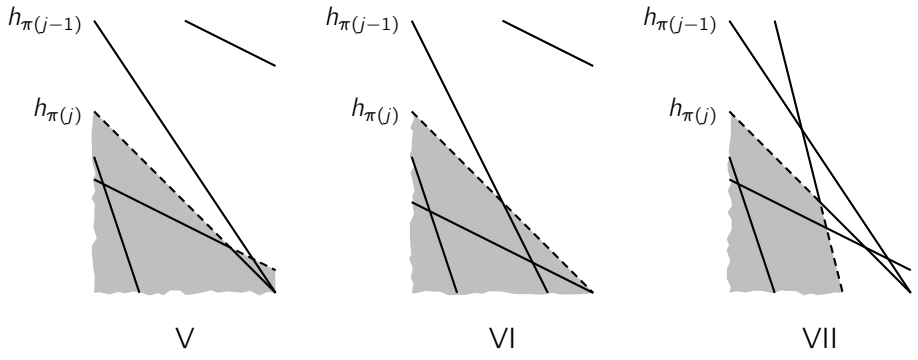


Figure 3.16: Three possibilities for the next intersection point of the current critical hyperplane $h_{\pi(j)}$ with (V) an infeasible hyperplane, (VI) its successor $h_{\pi(j-1)}$, (VII) another feasible hyperplane.

Including these amendments in Algorithm 3.2, the adapted version can compute all supported solutions of (3o$_{pnn}$.1c) corresponding to the feasible area. The corresponding supported points can easily be computed as well. This approach corresponds to Steps 1 and 2 of Algorithm 3.6 and can be seen as a preprocessing algorithm. In Steps 3 and 4 of

Algorithm 3.6, any of the approaches described in Section 3.4.1 can be applied to compute the remaining extreme supported points of the infeasible area.

Example 3.23 *Figure 3.17 presents the full decomposition of the projected weight space corresponding to Example 3.22. In comparison to Figure 3.14, the cells associated to* $(1,0,0,0,0)^\top$, $(0,1,0,0,0)^\top$ *and* $(0,1,0,1,0)^\top$ *extend and fill most of the infeasible area. The only new solution is* $(0,1,0,1,1)^\top$*, which is the lexicographic maximal solution for the first objective function.*

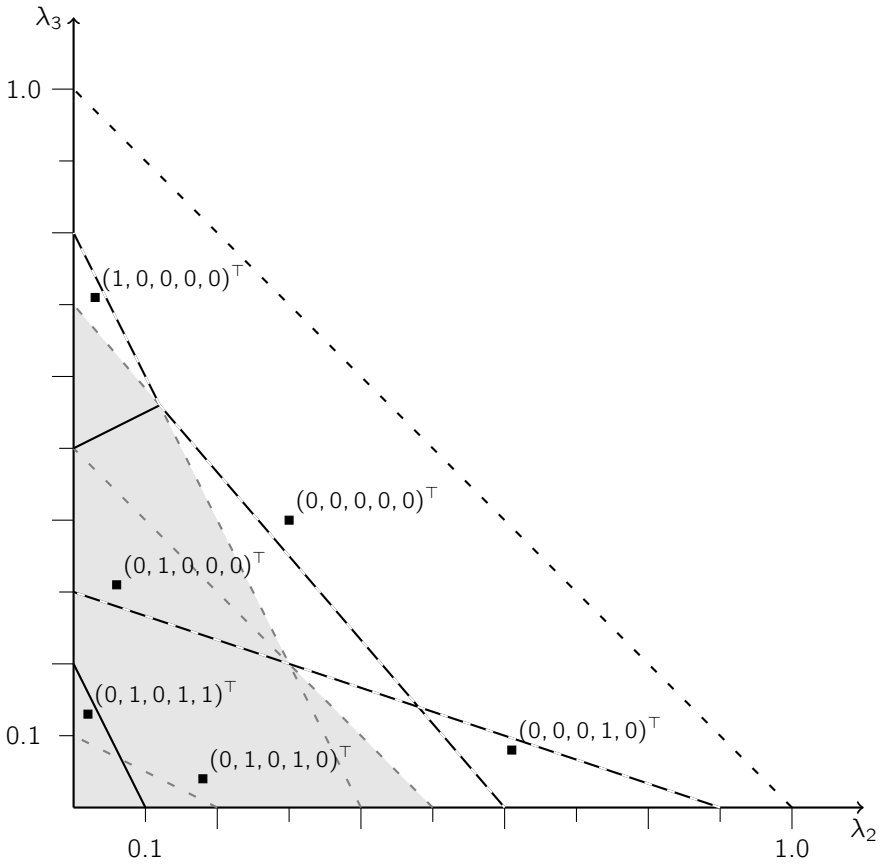


Figure 3.17: Full weight space decomposition corresponding to Example 3.22. The arrangement of hyperplanes associated to the unconstrained instance is marked by dashed lines and the infeasible area in light gray.

Computational results The above described approach was implemented in C++. The dichotomic search algorithm (Steps 3 and 4 of Algorithm 3.6) was coded using the approach of Przybylski et al. [2017]. The classical knapsack problem (1o.1c), which arises when solving the weighted sum problems, were solved using the MINKNAP algorithm of Pisinger [2015] [see also Pisinger, 1997]. The experiments were performed on an Intel Quadcore 2,80 GHz with 4 GB RAM.

Instances were generated following the described scheme with positive coefficients in the first objective function and the constraint, and negative coefficients in the second and third objective function. The absolute values of the coefficients were randomly chosen in the interval $[1, 10n]$. Three different values for the slackness c of the constraint, where $W = c \cdot \sum_{i=1}^{n} w_i$, were tested. Instances with large value for the slackness usually allow a larger percentage of solutions that are not influenced by the constraint as compared to instances with a small slackness value. Instances with 100 up to 1000 variables were tested. The average solution times and numbers of extreme supported points over 30 instances are presented in Table 3.5.

		time in s			$\lvert\mathcal{X}_{sE}\rvert$		
c	n	A	PP	DS	A	PP	DS
0.25	100	0.077	0.016	0.061	657.5	439.0	218.5
	200	0.783	0.087	0.696	2 554.7	1 737.9	816.8
	300	2.917	0.289	2.628	5 258.1	3 626.4	1 631.7
	400	8.397	0.998	7.399	9 691.0	6 949.7	2 741.4
	500	18.844	2.469	16.375	14 575.0	10 544.2	4 030.7
	750	100.089	12.103	87.986	32 272.8	23 405.5	8 867.3
	1000	414.929	45.000	369.929	57 588.3	42 788.1	14 800.2
0.50	100	0.073	0.033	0.040	1 224.3	1 094.8	129.4
	200	0.954	0.349	0.605	4 969.0	4 459.2	509.8
	300	4.449	1.789	2.660	10 633.2	9 531.1	1 102.1
	400	13.254	6.130	7.124	19 058.7	17 302.8	1 755.9
	500	31.126	15.185	15.941	29 233.6	26 678.7	2 554.9
	750	194.792	80.068	114.724	66 361.0	60 630.6	5 730.3
	1000	879.887	314.798	565.089	117 892.7	108 505.5	9 387.3
0.75	100	0.061	0.049	0.012	1 552.0	1 511.3	40.7
	200	0.870	0.652	0.218	6 443.2	6 289.8	153.4
	300	4.413	3.469	0.944	13 951.2	13 629.1	322.1
	400	14.494	11.757	2.737	25 113.5	24 578.5	535.0
	500	34.734	28.822	5.912	38 529.5	37 788.9	740.6
	750	209.799	158.434	51.365	87 614.3	86 087.6	1 526.7
	1000	925.996	692.988	233.008	155 924.2	153 257.2	2 667.0

Table 3.5: CPU-times, number of supported points and time per point for instances of ($3o_{pnn}$.1c) with 100 up to 1000 items (each averaged over 30 instances). The columns show the results for the overall algorithm (A), and the partial results for the preprocessing (PP) and the dichotomic search (DS).

Due to the fact that, on the one hand, the knapsack solver needs integer values as input data, but on the other hand the weight coefficients may become too large if expanded to values in $\mathbb{N}$, the coefficients may have to be rounded. Thus, numerical instabilities may occur and the dichotomic search may miss some extreme supported points. However, the

results are still very clear and the scope is to test the preprocessing algorithm, which is exact. Therefore, we keep the solver of Pisinger [1997].

The results show that most of the extreme supported points belong to the feasible part of the arrangement of hyperplanes and can be computed by the preprocessing (PP), even for instances with a slackness $c = 0.25$. In contrast, most of the CPU-time is spent for the dichotomic search (DS), which confirms that it is useful to apply the preprocessing algorithm.

		DS with PP		DS without PP	
c	n	time in s	$\lvert\mathcal{X}_{sE}\rvert$	time in s	$\lvert\mathcal{X}_{sE}\rvert$
0.25	100	0.077	657.5	0.252	657.5
	200	0.783	2 554.7	3.216	2 554.5
	300	2.917	5 258.1	13.099	5 256.1
	400	8.397	9 691.0	49.426	9 684.7
	500	18.844	14 575.0	136.275	14 561.4
	750	100.089	32 272.8	809.173	32 184.2
	1000	414.929	57 588.3	2 723.480	57 356.0
0.50	100	0.073	1 224.3	0.777	1 224.0
	200	0.954	4 969.0	11.477	4 968.2
	300	4.449	10 633.2	62.492	10 626.7
	400	13.254	19 058.7	255.941	19 038.0
	500	31.126	29 233.6	659.510	29 190.0
	750	194.792	66 361.0	3 617.104	66 084.5
	1000	879.887	117 892.7	11 501.604	117 176.4
0.75	100	0.061	1 552.0	1.203	1 551.7
	200	0.870	6 443.2	19.406	6 441.9
	300	4.413	13 951.2	123.193	13 941.3
	400	14.494	25 113.5	474.698	25 081.2
	500	34.734	38 529.5	1 189.565	38 461.1
	750	209.799	87 614.3	6 404.522	87 186.5
	1000	925.996	155 924.2	20 088.693	154 820.2

Table 3.6: CPU-times and number of supported points for instances of ($3o_{pnn}$.1c) with 100 up to 1000 items (each averaged over 30 instances) computed by a dichotomic search including the preprocessing as described above (DS with PP) and by a pure dichotomic search algorithm (DS without PP).

For comparison, we also computed the set of extreme supported points for all instances using solely the dichotomic search algorithm (DS without PP) of Przybylski et al. [2017]. The results are presented in Table 3.6 and are compared to those achieved by including the preprocessing (DS with PP). We observe that the number of extreme supported points differs depending on the applied approach. This can be explained by the numerical instabilities, which of course have more influence when solely using the dichotomic search

algorithm. The results again clearly show that the preprocessing considerably speeds up the process.

3.5 Conclusion and further ideas

In this chapter we analyzed the interrelation between the multi-objective unconstrained combinatorial optimization problem (mo.0c), zonotopes, arrangements of hyperplanes, and weight space decompositions. We showed that the convex hull of feasible points $\text{conv}(\mathcal{Y})$ can be defined by a zonotope and that the corresponding weight space decomposition is built by an arrangement of hyperplanes. As a consequence, it can be stated that each extreme supported point is generated by exactly one extreme supported solution and that the number of extreme supported solutions is bounded by $2 \cdot \sum_{i=0}^{m-1} \binom{n-1}{i}$. Hence, for a fixed number of objectives m, (mo.0c) has at most $\mathcal{O}(n^{m-1})$ extreme supported solutions. The observation of Visée et al. [1998] of a polynomial number of supported points for bi-objective knapsack problems can be proved for (mo.0c).

We showed that the structure of the arrangement of hyperplanes in the weight space allows an efficient computation of the extreme supported solutions of (mo.0c). We presented computational results for tri-objective problems with positive coefficients in the first and negative coefficients in the second and third objective function. The algorithm runs fast and reflects the theoretical results. We described how the algorithm can be generalized to arbitrary weights and to higher dimensions. As a future goal we want to implement algorithms for these cases and test the limits of the approach.

We demonstrated that the computation of extreme supported solutions for (mo.0c) can be used as a preprocessing for dichotomic search algorithms for (mo.1c). Our numerical study for ($3o_{pnn}$.1c) reveals that this approach considerably speeds up the process of computing all extreme supported points. Future research should address an extension to (mo.dc) with positive coefficients in the constraints and, in a second step, to (mo.dc) including arbitrary integer coefficients in the constraints. This extension would naturally continue our studies. We conjecture that the computation of extreme supported points of the corresponding (mo.0c) problem would still be a useful preprocessing.

Furthermore, it is an interesting question whether the results on (mo.0c) can be used also for other classes of (MOCO) problems. As Seipp [2013] shows, arrangements of hyperplanes also appear in the context of multi-objective minimum spanning tree problems. So there may be even more (MOCO) problems that have structural properties corresponding to concepts of combinatorial geometry.

A further idea for future research is to combine the results of this chapter with a second phase algorithm to compute the whole set of nondominated points for (mo.0c) and (mo.1c). It is an interesting question whether the interrelation of (mo.0c) and zonotopes reveals new insights also in the context of unsupported points. However, this is a challenging task since both problems are, in general, intractable.

4 Bi-dimensional knapsack problems with a soft constraint

In this chapter we present a bi-objective dynamic programming approach to solve a bi-dimensional knapsack problem. From an application point of view, on the one hand, some of the constraints in multi-dimensional knapsack problems may be hard in the sense that any violation, even a very minor one, is not acceptable. On the other hand, some other constraints may be soft or even uncertain, and constraint violations may be acceptable if the trade-off with respect to the potential improvements in the objective functions is favorable. Conversely, it may be interesting to reduce the capacity of one constraint even if this results in a reduction of the objective function value, as long as the trade-off is favorable.

In this case, a sensitivity analysis on the right-hand side values of the soft constraints provides alternative solutions that may be interesting to a decision-maker. For (1o.1c), one may assume that *adjacent problems*, i. e., (1o.1c) instances where only the right-hand sides differ and they differ by 1, have the same or at least similar optimal solutions. Blair [1998] shows that, even though this seems to be often the case, it cannot be expected in general. Woeginger [1999] proves a conjecture of Blair [1998] stating that already the decision problem asking whether the optimal solutions of two adjacent (1o.1c) problems share at least one selected item is $\mathcal{NP}$-complete. Even worse so, Blair [1998] shows that for any pair of (1o.1c) instances, two adjacent problems (1o.1c) of larger size can be formulated that have the optimal solutions of the initial problems. Hence, all (1o.1c) with differing right-hand side values have to be solved individually in general.

We follow a different approach in this chapter: Soft constraints are relaxed and re-interpreted as additional objective functions in a bi- or multi-objective model. The multi-objective perspective provides a whole set of solution alternatives, including the optimal solution of the multi-dimensional problem and additional solutions which are in a sense close in the objective space. Accordingly, the goal of this chapter is to propose a bi-objective approach for bi-dimensional knapsack problems with one soft constraint by adapting a bi-objective algorithm to the transformed problem. The additional computational effort for providing additional information is evaluated. It is shown that in practice this is an efficient procedure to generate solution alternatives and trade-off information.

In a more general context, the close relation between constrained optimization problems and multi-objective optimization problems is discussed in Klamroth and Tind [2007] [see also Gorski, 2010]. From an algorithmic point of view, solution concepts originally

developed for multi-objective problems can in this way be adapted for multi-constrained problems and vice versa. For example, this is successfully implemented in the case of constrained shortest path problems in Lozano and Medaglia [2013] and in the case of weight constrained minimum spanning tree problems in Henn [2007] [see also Ruzika, 2008]. Beier and Vöcking [2006] reformulate knapsack problems (1o.1c) into bi-objective unconstrained combinatorial optimization problems (2o.0c) and combine bi-objective dynamic programming with the core concept to solve (1o.1c).

In this chapter, the special case of the bi-dimensional knapsack problem is analyzed and an associated bi-objective KP is formulated that has one maximization (profit) and one minimization objective (weight or cost). We study the relationship between these two problems from a theoretical as well as from an experimental perspective. We adapt the DP approach for bi-objective knapsack problems of Figueira et al. [2013] to the case of one maximization and one minimization objective to produce the exact solution of the bi-dimensional problem and, in addition, interesting solution alternatives, providing trade-off information between profit optimization and constraint satisfaction. Since the DP algorithm determines all alternative solutions corresponding to equal values in the objective functions, this provides even more information to a decision-maker.

The chapter is organized as follows: In Section 4.1 the transformation between the bi-dimensional and bi-objective knapsack problem is defined and justified. The general structure of the bi-objective DP approach is described in Section 4.2. The preprocessing algorithm and the pruning strategies that are applied during the DP algorithm are presented in Section 4.3. This includes known dominance relations that are adapted to the structure of the problem and bounds that can be introduced due to this structure. In Section 4.4 we discuss the adaption of the DP algorithm for computing a predefined subset of the nondominated solutions. Section 4.5 reports computational experiments and the corresponding results. We reconsider our ideas in the context of multi-dimensional knapsack problems in Section 4.6. The results of Chapter 3 play an important role for computing extreme supported points in a first phase for higher dimensions. For (2o.1c) with only two objectives, the dichotomic search algorithm is very fast such that we do not use the algorithms of Chapter 3. Conclusion and avenues for future research are presented in Section 4.7.

4.1 Reformulating constraints as objectives

Let a bi-dimensional knapsack problem (1o.2c) be given:

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} p_i^1 x_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} w_i^k x_i \leq W^k, \qquad k = 1, 2 \\ & x_i \in \{0, 1\}, \qquad i = 1, \ldots, n. \end{aligned} \tag{1o.2c}$$

We assume that the first constraint with weight coefficients w_i^1 and capacity W^1 is a hard constraint, whereas the second constraint with weight coefficients w_i^2 and capacity W^2 is a soft constraint. Below we continue using the terms *hard* and *soft constraint*, respectively, to distinguish between them. Following the idea of computing several interesting solution alternatives, the soft constraint is transformed and reinterpreted as an additional objective function that is to be minimized. By altering the sign of the weight coefficients p_i^2 such that $p_i^2 := -w_i^2 < 0$, for $i = 1, \ldots, n$, we obtain a bi-objective knapsack problem ($2o_{pn}$.1c) with both maximization objectives:

$$\begin{aligned} \text{vmax} \quad & f(x) = (f_1(x), f_2(x)) = \left(\sum_{i=1}^{n} p_i^1 x_i, \sum_{i=1}^{n} p_i^2 x_i \right) \\ \text{s. t.} \quad & \sum_{i=1}^{n} w_i^1 x_i \leq W^1 \\ & x_i \in \{0,1\}, \quad i = 1, \ldots, n. \end{aligned} \tag{$2o_{pn}$.1c}$$

For convenience, we refer to f_1 as the *original objective function* and to f_2 as the *transformed objective function*, respectively. In this chapter, ($2o_{pn}$.1c) denotes the bi-objective knapsack problem where all coefficients in the first objective function are positive and all coefficients in the second objective function are negative.

By applying the transformation from (1o.2c) to ($2o_{pn}$.1c) we are interested in generating several alternative solutions to an optimal bi-dimensional solution x^* of (1o.2c). In particular, x^* is an element of the set of weakly efficient solutions of ($2o_{pn}$.1c) and thus we do not loose any information by the transformation.

Indeed, from a multi-objective perspective (1o.2c) can be seen as an ε-constraint scalarization of ($2o_{pn}$.1c), with f_2 reversely transformed into a constraint and the corresponding bound value ε set to W^2. Chankong and Haimes [1983] showed that every optimal solution of (1o.2c) is weakly efficient for ($2o_{pn}$.1c), and at least one of the optimal solutions of (1o.2c) is efficient for ($2o_{pn}$.1c) (see also Section 2.1). Therefore, if there is a unique optimal solution of (1o.2c), it is an efficient solution of ($2o_{pn}$.1c) and the set of efficient solutions of ($2o_{pn}$.1c) contains at least one optimal solution of (1o.2c).

In other words, if the nondominated set $\mathcal{Y}_N$ of ($2o_{pn}$.1c) is known, an optimal solution of (1o.2c) is given by:

$$x^* = \arg\max_{x \in \mathcal{X}} \{ f_1(x) \,:\, f(x) \in \mathcal{Y}_N,\ f_2(x) \geq -W^2 \}.$$

4.2 Dynamic programming algorithm

DP algorithms are based on implicit enumeration (for a general introduction see Section 2.2). In the case of (2o.1c), the procedure is split into n steps, called *stages*. Each stage S_k, $k \in \{1, \ldots, n\}$, contains *states* $s = (s_1, s_2, s_3)$ corresponding to feasible solutions of problem (2o.1c) and their images in the objective space. In particular, we assume

that all states in a stage S_k, for $k = 1, \ldots, n$, correspond to partial solutions $x \in \{0,1\}^n$ in the sense that $x_{k+1} = \ldots = x_n = 0$. If a solution $x \in \mathcal{X}$ corresponds to a state s this means that $s_1 = f_1(x)$, $s_2 = f_2(x)$ and s_3 equals the value of the hard constraint, i. e., $s_3 = \sum_{i=1}^{n} w_i^1 x_i$. In the following, we use the notion of dominance also for states: A state s is dominated by another state $\hat{s}$ if and only if the corresponding solution x is dominated by $\hat{x}$. More precisely, s is dominated by $\hat{s}$ if and only if

$$\hat{s}_1 \geq s_1, \quad \hat{s}_2 \geq s_2, \quad \text{and} \quad (\hat{s}_1, \hat{s}_2) \neq (s_1, s_2).$$

Moreover, new stages S_k are created by adding (1-extension) or not adding (0-extension) the coefficients p_k^1, p_k^2 and w_k^1 of item k to the values s_1, s_2 and s_3 of every state s in stage S_{k-1}, respectively. This means that item k is added to the partial solution of the previous stage, or not. The 1-extensions are only allowed if the resulting value $\hat{s}_3 = s_3 + w_k^1$ is smaller than or equal to the capacity W^1, i. e., if the corresponding solution stays feasible. In the following we often analyze a state $s \in S_k$, for $k = 1, \ldots, n-1$, and all of its extensions, hence we define the *set of feasible extensions of* s:

$$\text{ext}(s) = \Big\{ e = (e_1, e_2, e_3) : e_1 = s_1 + \sum_{i \in I} p_i^1, e_2 = s_2 + \sum_{i \in I} p_i^2, e_3 = s_3 + \sum_{i \in I} w_i^1, \\ e_3 \leq W^1, I \subseteq \{k+1, \ldots, n\} \Big\}.$$

The overall process starts with the initial stage $S_0 = \{(0,0,0)\}$ in which no item has been selected and no item has been considered yet. The states of the last stage S_n correspond to the complete set of feasible points $\mathcal{Y}$. The corresponding solutions can be determined using standard bookkeeping techniques.

A central idea in dynamic programming is to make use of Bellman's *Principle of Optimality* (see Section 2.2). We are only interested in efficient solutions and, corresponding to that, in the nondominated set $\mathcal{Y}_N$. We can thus prune states of the DP process that only produce dominated extensions. The applied pruning strategies, named dominance relations, are described in Section 4.3. These dominance relations, *Dom*, are applied in a recursive way to the set of newly generated states in stage S_k based on stage S_{k-1}. Then the last stage S_n corresponds to $\mathcal{Y}_N$ and all efficient solutions, including alternative solutions corresponding to the same nondominated point, can be determined.

Summarizing the discussion above the following recursion is applied for $k = 1, \ldots, n$, starting with $S_0 = \{(0,0,0)\}$:

$$S_k = \text{Dom}\Big(S_{k-1} \cup \big\{ (s_1 + p_k^1, s_2 + p_k^2, s_3 + w_k^1) : s_3 + w_k^1 \leq W^1, s \in S_{k-1} \big\} \Big).$$

Figure 4.1 illustrates a DP process, which can be described as a network without directed cycles. A node is introduced for every pair of a stage S_k, $k = 0, \ldots, n$, and a realized weight value w^1, $0 \leq w^1 \leq W^1$. Therefore, several states can be allocated to one node, see Figure 4.1. Edges are connecting nodes of consecutive states, where a state allocated to a node in S_k has to be an extension of a state allocated to the node in S_{k-1}, for $k = 1, \ldots, n$. We thus use the term *DP network* in the following.

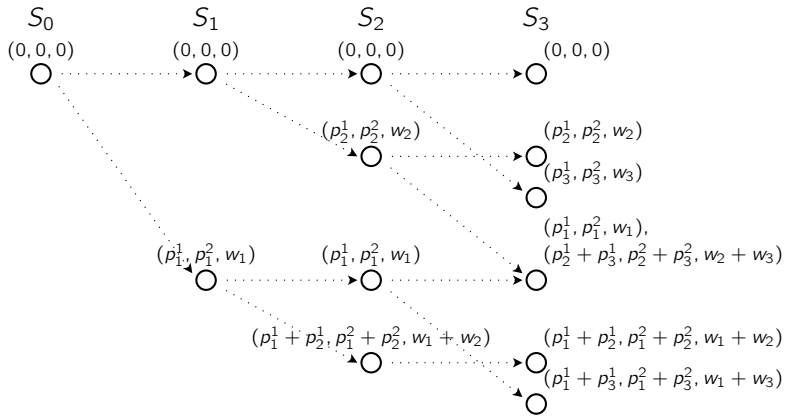


Figure 4.1: Example of Dynamic Programming network for problem (2o.1c) where in this example we assume that $w_1^1 = w_2^1 + w_3^1$ and that $w_1^1 + w_2^1 + w_3^1 > W^1$.

4.3 Preprocessing and pruning strategies

The general DP approach described above can be applied independently from the signs of the coefficients. But it makes a difference for the applicability of pruning strategies. Thus, the negative coefficients in the second objective function of (2o$_{pn}$.1c) need special attention. In Section 4.3.2, two dominance relations DominanceA/B and UpperBound are presented that were introduced in Figueira et al. [2013] and are adapted here to the case of negative objective coefficients. In Section 4.3.3, an upper bound procedure SearchZones is introduced that is based on the occurrence of negative coefficients.

The set of extreme supported points $\mathcal{Y}_{eN}$ has to be precomputed to apply the pruning strategies. The preprocessing algorithm (Preprocessing) is done by weighted sum scalarizations on (2o$_{pn}$.1c). Certainly, the results of Chapter 3 can be applied to compute $\mathcal{Y}_{eN}$. However, in the bi-objective case and for the number of items in our computational tests, the applied dichotomic search algorithm is very fast and, thus, we do not use the precomputation of extreme supported solution based on the unconstrained version of (2o$_{pn}$.1c). The preprocessing algorithm is described in detail in Section 4.3.1. We assume without loss of generality that the extreme supported points $\{\bar{s}^1, \ldots, \bar{s}^q\} = \mathcal{Y}_{eN}$ are sorted in increasing order of values of the original objective function f_1.

Algorithm 4.1 gives a pseudocode of the DP procedure for problem (2o$_{pn}$.1c). We use the three different dominance relations (DominanceA/B, UpperBound, and SearchZones) and Preprocessing to compute $\mathcal{Y}_{eN}$. Branch generates the candidate set for the following stage on which the different dominance relations are performed.

4.3.1 Bi-objective dichotomic search

In the bi-objective case, the dichotomic search algorithm of Aneja and Nair [1979] can be applied to analyze the weight space; see Algorithm 4.2, cf. Sections 2.1 and 3.4.1. First, the two lexicographic maxima x_1 and x_2 are computed. All further nondominated points

Algorithm 4.1 DP algorithm for (2o$_{pn}$.1c)

Input: n, $\mathcal{P}^1 = \{p_1^1, \ldots, p_n^1\}$, $\mathcal{P}^2 = \{p_1^2, \ldots, p_n^2\}$, $\mathcal{W} = \{w_1^1, \ldots, w_n^1\}$, W^1.
1: $\mathcal{Y}_{eN} :=$ PREPROCESSING($\mathcal{P}^1$, $\mathcal{P}^2$, $\mathcal{W}$, W^1)
2: $S := \{(0, 0, 0)\}$
3: **for** $k := 1, \ldots, n$ **do**
4: $\quad S :=$ BRANCH(S)
5: $\quad$ **if** $k = n$ **then**
6: $\quad\quad \mathcal{Y}_N :=$ DOMINANCEB(S)
7: $\quad$ **else**
8: $\quad\quad S :=$ DOMINANCEA(S)
9: $\quad\quad S :=$ UPPERBOUND($\mathcal{P}^1$, $\mathcal{W}$, W^1, S, $\mathcal{Y}_{eN}$)
10: $\quad\quad S :=$ SEARCHZONES($\mathcal{P}^1$, $\mathcal{P}^2$, $\mathcal{W}$, W^1, S, $\mathcal{Y}_{eN}$)
Output: $\mathcal{Y}_N$

will lie within a *search zone* defined by the corresponding lexicographic points $f(x_1)$ and $f(x_2)$ as follows:

$$\mathcal{C}(f(x_2), f(x_1)) = \big(f_1(x_2), f_2(x_1)\big) + \mathbb{R}^2_{\geqq}.$$

Note that $\mathcal{C}(f(x_2), f(x_1))$ defines a cone in $\mathbb{R}^2$. Then, the algorithm computes a weighted sum objective defined by the weights (λ_1, λ_2) defined as

$$\lambda_1 := f_2(x_2) - f_2(x_1) \quad \text{and} \quad \lambda_2 := f_1(x_1) - f_1(x_2).$$

The optimal solution $\bar{x}$ of the resulting single-objective knapsack problem (1o.1c) corresponds to a supported point of (2o$_{pn}$.1c).

The search zone $\mathcal{C}(f(x_2), f(x_1))$ can be split into two new search zones $\mathcal{C}(f(x_2), f(\bar{x}))$ and $\mathcal{C}(f(\bar{x}), f(x_1))$ using the new supported point $f(\bar{x})$. The procedure of solving the weighted sum problem for a search zone $\mathcal{C}(f(x_\alpha), f(x_\beta))$ and splitting it into two new ones can be applied iteratively. Note that the definition of the cones gives an ordering of the solutions: For $\mathcal{C}(f(x_\alpha), f(x_\beta))$ we know that $f_1(x_\alpha) < f_1(x_\beta)$ and that $f_2(x_\alpha) > f_2(x_\beta)$. We say that x_α corresponds to the left and x_β to the right supported point defining $\mathcal{C}(f(x_\alpha), f(x_\beta))$. If no new supported point is generated, i. e., if the weighted sum objective function value of the newly generated solution $\bar{x}$ is equal to the weighted sum objective function values of one of the solutions x_α, x_β, then there exists no extreme supported point of (2o$_{pn}$.1c) in the interior of $\mathcal{C}(f(x_\alpha), f(x_\beta))$. The search zone $\mathcal{C}(f(x_\alpha), f(x_\beta))$ can be discarded in this case (Step 13 in Algorithm 4.2). The solution x_α, which corresponds to the left supported point defining the search zone, is included in the set E. In this way, all computed solutions are included in E at some point during the course of the algorithm since all solutions (except the lexicographic maximal solution x_1 which is saved beforehand) correspond to the left supported point for exactly one (discarded) search zone.

Note that supported but nonextreme solutions may not be detected because nonextreme supported solutions have the same objective function value in the weighted sum scalariza-

tion as the solutions x_α and x_β which define $\mathcal{C}(f(x_\alpha), f(x_\beta))$. Therefore, it is guaranteed that all extreme supported points are computed but no statement can be made about the nonextreme supported solutions.

To summarize the discussion above, in a search zone $\mathcal{C}(f(x_\alpha), f(x_\beta))$ either a new supported point is found and two new search zones are generated or there exists no extreme point in $\mathcal{C}(f(x_\alpha), f(x_\beta))$ and this search zone is deleted. Note that exactly one knapsack problem has to be solved for each search zone. Since the number of supported points is finite, the procedure stops after a finite number of iterations, computing the set $\mathcal{Y}_{eN}$ and probably some additional nonextreme supported points.

To reduce the computational effort, the dichotomic search can be stopped after a fixed number of iterations. In this case, the dominance relations are applied using only a subset of $\mathcal{Y}_{eN}$. It is also possible to start the DP algorithm (Algorithm 4.1) with an arbitrary approximation of $\mathcal{Y}_{eN}$, which can be computed with a predefined time limit. However, this would in general lead to a weaker performance of the dominance relations. In our numerical tests we always executed the complete dichotomic search since this turned out to be very fast in practice.

Algorithm 4.2 Bi-objective dichotomic search [Aneja and Nair, 1979]

Input: coefficients p_i^1, p_i^2, w_i^1, for $i = 1, \ldots, n$, W^1.

compute lexicographic maximal solutions x_1, x_2 with respect to $f_1(x)$, $f_2(x)$
$f(x_1) := (f_1(x_1), f_2(x_1))$, $f(x_2) := (f_1(x_2), f_2(x_2))$, and $E := \{x_2\}$
if $f(x_1) \neq f(x_2)$ **then**
    $L := \{\mathcal{C}(f(x_2), f(x_1))\}$ // *list of search zones*
    $\ell := 1$
    while $\ell \geq 1$ **do**
        select $\mathcal{C}(f(x_\alpha), f(x_\beta)) \in L$
        $\lambda_1 := f_2(x_\alpha) - f_2(x_\beta)$ and $\lambda_2 := f_1(x_\beta) - f_1(x_\alpha)$
        for $i := 1, \ldots, n$ **do**
            $\bar{p}_i := \lambda_1 p_i^1 + \lambda_2 p_i^2$
        define (1o.1c) with profits $\bar{p}_i$ and weights w_i^1, $i = 1, \ldots, n$
        compute optimal solution $\bar{x}$ of (1o.1c)
        if $(\lambda_1 f_1(\bar{x}) + \lambda_2 f_2(\bar{x})) = (\lambda_1 f_1(x_\alpha) + \lambda_2 f_2(x_\alpha))$ **then**
            $E := E \cup \{x_\alpha\}$
            $L := L \setminus \{\mathcal{C}(f(x_\alpha), f(x_\beta))\}$
            $\ell := \ell - 1$
        else
            $L := L \cup \{\mathcal{C}(f(x_\alpha), f(\bar{x})), \mathcal{C}(f(\bar{x}), f(x_\beta))\} \setminus \{\mathcal{C}(f(x_\alpha), f(x_\beta))\}$
            $\ell := \ell + 1$

Output: E // *E corresponds to U with $\mathcal{Y}_{eN} \subseteq U \subseteq \mathcal{Y}_{sN}$, i.e., to a subset U of $\mathcal{Y}_{sN}$, which contains the complete set $\mathcal{Y}_{eN}$*

4.3.2 Dominance relations DominanceA/B and UpperBound

In Bazgan et al. [2009b] three different dominance relations for (2o$_{pn}$.1c) are proposed, which are referred to as (**D1**), (**D2**) and (**D3**) in Figueira et al. [2013]. While the dominance relation (**D1**) cannot be adapted to negative coefficients, relations (**D2**) and (**D3**) turn out to be useful in the following.

The dominance relation (**D1**) cannot be adapted for the following reason: It is based on testing if the currently regarded item k and all of the remaining items $k+1, \ldots, n$ fit into the knapsack. Having only positive coefficients, both objective functions improve by including items. In this case the state resulting from the 0-extension can be discarded if the complete 1-extension, i. e., adding all items $k, \ldots, n$, is feasible. Since in our case one objective function with negative coefficients is maximized, this is no longer true, and the 0-extension may still produce nondominated states even if all remaining items can be added to the partial solution at hand.

The two remaining dominance relations can also be applied in the case of negative coefficients. In (**D2**), or DominanceA and B in Algorithm 4.1, a state $s \in S_n$ can be discarded, if there exists another state $\hat{s} \in S_n$, $s \neq \hat{s}$, and s is dominated by $\hat{s}$. For all stages S_k with $k < n$ the weights s_3 and $\hat{s}_3$ have to be considered because the DP algorithm can add more of the remaining items to a solution corresponding to a state with a lower weight. So, if s is dominated by $\hat{s}$ and $s_3 \geq \hat{s}_3$ then every extension $e \in \text{ext}(s)$ is dominated by at least one of the extensions in $\text{ext}(\hat{s})$. Thus, s can be discarded. But if $s_3 < \hat{s}_3$ it is still possible that one of the extension $e \in \text{ext}(s)$ is not dominated by any extension $\hat{e} \in \text{ext}(\hat{s})$. State s cannot be discarded in this case. DominanceA in Algorithm 4.1 considers the values s_1, s_2 and s_3. If $k = n$, DominanceB is used, which only compares the first and second objective function values s_1 and s_2.

The third proposed dominance relation UpperBound [see (**D3**) with variant B-DP1 in Figueira et al., 2013] uses an upper bound $u(s) = (u_1(s), u_2(s))$ on all possible extensions of s, i. e., $e_1 \leq u_1(s)$ and $e_2 \leq u_2(s)$ for every $e \in \text{ext}(s)$. If $u(s)$ is already dominated by one of the extreme supported points $\bar{s} \in \mathcal{Y}_{eN}$, then s can be discarded because neither s itself nor one of the extensions will be nondominated. Let $s \in S_k$, $k \in \{1, \ldots, n\}$. The state s can be discarded, if there exists a supported point $\bar{s}^j \in \mathcal{Y}_{eN}$, for $j \in \{1, \ldots, q\}$, with

$$\bar{s}_1^j \geq u_1(s), \quad \bar{s}_2^j \geq u_2(s) \quad \text{and} \quad \bar{s}^j \neq u(s).$$

We thus need an efficient strategy to compute upper bounds $u_1(s)$ and $u_2(s)$, which can be implemented as follows:

The upper bound in the original objective function $u_1(s)$ is computed according to the improved Martello and Toth bound [Martello and Toth, 1977] for the classical knapsack problem. It only uses the coefficients of f_1 and the hard constraint. The not yet considered items $k+1, \ldots, n$ are ordered non-increasing according to their efficiencies, i. e., their profit to weight ratios p_i^1 / w_i^1. According to this order the items are added into the knapsack until the break item is reached, i. e., until the first item would violate the constraint. The break item is identified by the index b (see Section 2.2.1). The residual capacity $\overline{W}^1$ is

calculated as follows:

$$\overline{W}^1 := W^1 - s_3 - \sum_{j=k+1}^{b-1} w_j^1.$$

To obtain an upper bound on f_1, the integrality constraint is relaxed for one item, but not for the break item b, to get equality in the constraint. There are two possibilities for the optimal solution of this partially relaxed knapsack problem: The break item is included or not. We consider all items $1, \ldots, b-1$ together with item b or $b+1$, respectively. Either item b is included, at the cost of removing a corresponding multiple $(w_b^1 - \overline{W}^1)/w_{b-1}^1$ of item $b-1$ (note that this multiple could be larger than 1), or the corresponding multiple $\overline{W}^1/w_{b+1}^1$ of item $b+1$ is used to fill the remaining capacity. The maximum of both results is an upper bound on the first objective value. Additionally, the assumption that all data is integer allows to round this value down to the next integer:

$$u_1(s) = s_1 + \sum_{j=k+1}^{b-1} p_j^1 + \max\left\{ \left\lfloor p_b^1 - (w_b^1 - \overline{W}^1) \cdot \frac{p_{b-1}^1}{w_{b-1}^1} \right\rfloor, \left\lfloor \overline{W}^1 \cdot \frac{p_{b+1}^1}{w_{b+1}^1} \right\rfloor \right\}.$$

In the transformed objective function f_2 the upper bound $u_2(s)$ is set to the value s_2, i. e., $u_2(s) = s_2$. This is indeed an upper bound on the second objective since, with every additional item, the value of f_2, which is to be maximized, can only be reduced further.

4.3.3 Bounds induced by search zones

The upper bound $u_2(s)$ is in general not a strong bound. For every extension of s (that is not equal to s itself), the value of f_2 will become smaller. Additionally, we know that nondominated, but nonextreme points can only be in regions of the objective space that are not dominated by the extreme supported points in $\mathcal{Y}_{eN}$. These regions correspond to triangles, named *search triangles*, which are part of the search zone defined by a *local lower bound* [Klamroth et al., 2015, Dächert, 2014]: Let $\bar{s}^j$, $\bar{s}^{j+1}$, for $j \in \{1, \ldots, q-1\}$, be two consecutive extreme supported points from the set $\mathcal{Y}_{eN}$, i. e., $\bar{s}_1^j < \bar{s}_1^{j+1}$ and $\bar{s}_2^j > \bar{s}_2^{j+1}$. The point $(\bar{s}_1^j, \bar{s}_2^{j+1})$ defines a local lower bound for the corresponding search zone $\mathcal{C}(\bar{s}^j, \bar{s}^{j+1}) = (\bar{s}_1^j, \bar{s}_2^{j+1}) + \mathbb{R}^2_{\geq}$, for all $j = 1, \ldots, q-1$. The search triangles are triangles defined by the points $\bar{s}^j$, $\bar{s}^{j+1}$, and the local lower bound $(\bar{s}_1^j, \bar{s}_2^{j+1})$. There can be no nondominated points lying inside the search zone that are not lying in the search triangle, because those would be supported points.

The search triangles and corresponding local lower bounds are illustrated in Figure 4.2. In the following we investigate the regions $[\bar{s}_1^j, \bar{s}_1^{j+1}) \times \mathbb{Z}^-$, for all $j = 1, \ldots, q-1$, and $\{\bar{s}_1^q\} \times \mathbb{Z}^-$ (illustrated for $j = 2$ in Figure 4.2), which also include the corresponding search triangles. To simplify the notation, we introduce a dummy point

$$\bar{s}^{q+1} := \bar{s}^q + \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

and increase the last region to $[\bar{s}_1^q, \bar{s}_1^{q+1}) \times \mathbb{Z}^-$. For every region with $\bar{s}_1^{j+1} > s_1$ and $\bar{s}_1^j \leq u_1(s)$, for $j \in \{1, \ldots, q\}$, we introduce an upper bound $u_2^j(s)$. It is computed by counting the minimum number a_j of items which need to be added to obtain states inside the region. This number allows to compute a, maybe not realizable, minimum cost in f_2 to implement this step, which is an upper bound on the component e_2 of all extensions e of s. Regions $[\bar{s}_1^j, \bar{s}_1^{j+1}) \times \mathbb{Z}^-$ with $\bar{s}_1^{j+1} \leq s_1$ are not of interest because $e_1 \geq s_1 \geq \bar{s}_1^{j+1}$ for all $e \in \text{ext}(s)$, i. e., no extension can be inside these regions. The same is true for regions where the upper bound $u_1(s)$ is smaller than $\bar{s}_1^j$ because $e_1 \leq u_1(s) < \bar{s}_1^j$ for all $e \in \text{ext}(s)$. The resulting procedure is named SearchZones in Algorithm 4.1. To simplify the notation, we partition the set of extensions of s into q subsets.

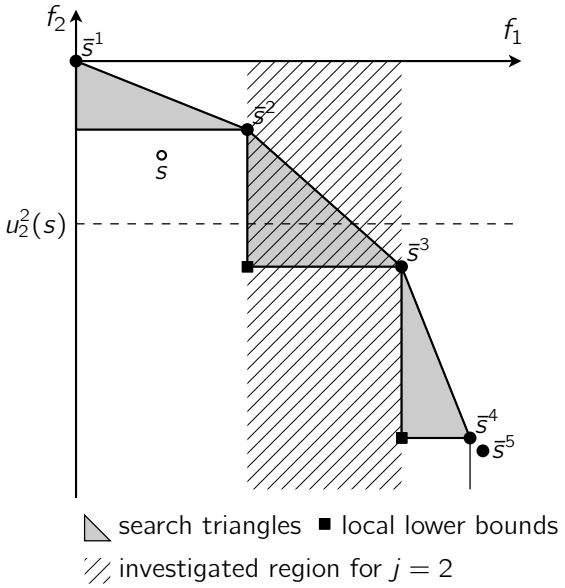


Figure 4.2: Illustration of search triangles, local lower bounds, the upper bound $u_2^2(s)$ for extensions e of state s with $\bar{s}_1^2 \leq e_1 < \bar{s}_1^3$, and the dummy point $\bar{s}^5$.

Definition 4.1 *Let $s \in S_k$, for $k \in \{1, \ldots, n\}$. For every extreme supported point $\bar{s}^j \in \mathcal{Y}_{eN}$, $j \in \{1, \ldots, q\}$, let $\text{ext}^j(s)$ be a subset of $\text{ext}(s)$ with:*

$$\text{ext}^j(s) = \left\{ e = (e_1, e_2, e_3) \in \text{ext}(s) \ : \ \bar{s}_1^j \leq e_1 < \bar{s}_1^{j+1} \right\}.$$

Remark 4.2 *Let $s \in S_k$, for $k \in \{1, \ldots, n\}$, with the notation of Definition 4.1.*

i) It holds that

$$\text{ext}(s) = \bigcup_{j \in \{1, \ldots, q\}} \text{ext}^j(s).$$

ii) Let $j \in \{1, \ldots, q\}$. If $\bar{s}_1^j > u_1(s)$ or if $\bar{s}_1^{j+1} \leq s_1$, then it holds that

$$\text{ext}^j(s) = \emptyset.$$

To compute upper bounds $u_2^j(s)$, we consider both objective functions separately. So we can use the best remaining items for each objective independently. To do so, we sort a subset of items in non-increasing order. For a given $k \in \{1, \ldots, n\}$, let $\mathcal{S}_{n-k}$ denote the symmetric group of order $n-k$ and $\pi_1,\ \pi_2 \in \mathcal{S}_{n-k}$ denote two permutation of the numbers $k+1, \ldots, n$, where $p^1_{\pi_1(k+1)} \geq \ldots \geq p^1_{\pi_1(n)}$ and $p^2_{\pi_2(k+1)} \geq \ldots \geq p^2_{\pi_2(n)}$.

Definition 4.3 *Let $s \in S_k$, for $k \in \{1, \ldots, n\}$. For $\bar{s}^j \in \mathcal{Y}_{eN}$, $j \in \{1, \ldots, q\}$, with $s_1 < \bar{s}_1^{j+1}$ and $u_1(s) \geq \bar{s}_1^j$ let*

$$a_j = \min_b \left\{ b \in \{k+1, \ldots, n\} : s_1 + \sum_{i=k+1}^{b} p^1_{\pi_1(i)} \geq \bar{s}_1^j \right\}.$$

For $j \in \{1, \ldots, q\}$, with $s_1 < \bar{s}_1^{j+1}$ and $u_1(s) \geq \bar{s}_1^j$, we define:

$$u_2^j(s) = s_2 + \sum_{i=k+1}^{a_j} p^2_{\pi_2(i)}.$$

The value $u_2^j(s)$ is an upper bound on the value of the second objective function for every extension of s which has a first objective function value greater than or equal to the value $\bar{s}_1^j$ of the corresponding extreme supported point $\bar{s}^j$. In particular, for all $j \in \{1, \ldots, q\}$ with $s_1 < \bar{s}_1^{j+1}$ and $u_1(s) \geq \bar{s}_1^j$, we have that $e_2 \leq u_2^j(s)$ for all $e \in \text{ext}^j(s)$. Now we can formulate the following theorem.

Theorem 4.4 *Let $s \in S_k$, for $k \in \{1, \ldots, n\}$, and let $\mathcal{J} = \left\{ j \in \{1, \ldots, q\} \ : \ s_1 < \bar{s}_1^{j+1} \right\} \cap \left\{ j \in \{1, \ldots, q\} \ : \ u_1(s) \geq \bar{s}_1^j \right\}$. If, for all $j \in \mathcal{J}$:*

$$u_2^j(s) \leq \bar{s}_2^{j+1} \tag{4.1}$$

then s itself is the only extension of s that can be nondominated, i. e., for all $e \in \text{ext}(s)$, $e \neq s$ it holds that $e \notin \mathcal{Y}_N$.

Proof. Let $s \in S_k$, for $k \in \{1, \ldots, n\}$, and $\mathcal{J} = \left\{ j \in \{1, \ldots, q\} \ : \ s_1 < \bar{s}_1^{j+1} \right\} \cap \left\{ j \in \{1, \ldots, q\} \ : \ u_1(s) \geq \bar{s}_1^j \right\}$. We assume that (4.1) holds for s for every $j \in \mathcal{J}$. Assume that there exists an extension $\widetilde{s} \in \text{ext}(s)$ with $\widetilde{s} \in \mathcal{Y}_N$.
Because $\widetilde{s} \in \mathcal{Y}_N$ the following statement holds:

($*$) It exists $j' \in \mathcal{J}$ such that $\widetilde{s} \in \text{ext}^{j'}(s)$ with

$$\bar{s}_1^{j'} \leq \widetilde{s}_1 < \bar{s}_1^{j'+1} \text{ and } \bar{s}_2^{j'} \geq \widetilde{s}_2 > \bar{s}_2^{j'+1}$$

for $\bar{s}^{j'}, \bar{s}^{j'+1} \in \mathcal{Y}_{eN} \cup \{\bar{s}^{q+1}\}$

We know that

- $\widetilde{s} \in \text{ext}(s) \ \Rightarrow \ \widetilde{s}_2 \leq u_2^{j'}(s)$

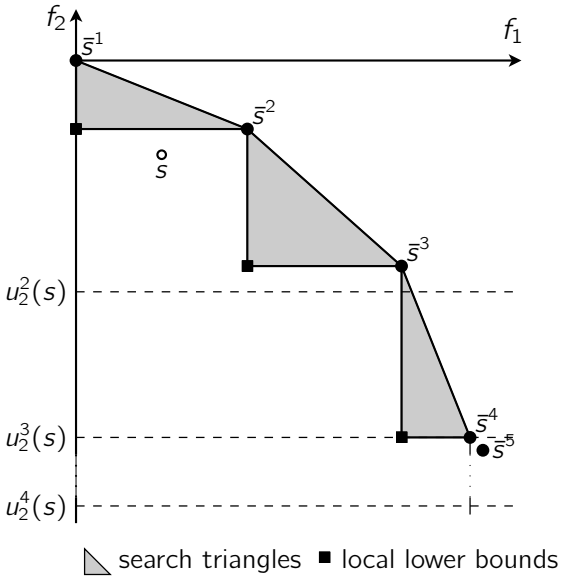


Figure 4.3: State s can be discarded: All upper bounds $u_2^j(s)$ are smaller than or equal to $\bar{s}_2^{j+1}$, $j \in \mathcal{J} = \{2, 3, 4\}$. Hence, all extensions $e \in \text{ext}(s)$, $e \neq s$, are dominated. In contrast, in Figure 4.2 the upper bound $u_2^2(s)$ is greater than $\bar{s}_2^3$ and hence extensions of s can be nondominated.

- $u_2^{j'}(s) \leq \bar{s}_2^{j'+1}$ because of (4.1)
- $\bar{s}_2^{j'+1} < \widetilde{s}_2$ because of $(*)$

Hence, $\widetilde{s}_2 < \widetilde{s}_2$, which is a contradiction. So $\widetilde{s}$ cannot be in $\mathcal{Y}_N$. □

If (4.1) holds for $s \in S_k$ with the corresponding set $\mathcal{J}$ (see Figure 4.3), the DP does not need to compute ext(s). Only s itself has to remain in the process. If this case occurs (especially at an early stage of the DP) the DP network is pruned significantly. None of the extensions of s has to be computed. The states $s \in S_n$ still correspond to $\mathcal{Y}_N$.

If, for one or more values of $j \in \mathcal{J}$, (4.1) is not true, s has to be extended further. But, it is not necessarily required to check all the regions for the extensions of s again, see Remark 4.5.

Remark 4.5 *Let $j \in \{1, \ldots, q\}$. If condition (4.1) is true for j for a state s, it is trivially true for j for all of the extensions $e \in \text{ext}(s)$ and does not need to be checked again.*

In practice, this could be used in the following way: If (4.1) is true for some $j \in \mathcal{J}$, j is deleted from $\mathcal{J}$. The extensions of s then need to be examined in $\widetilde{\mathcal{J}} = \left\{j \in \{1, \ldots, q\} \ : \ s_1 < \bar{s}_1^{j+1}\right\} \cap \left\{j \in \{1, \ldots, q\} \ : \ u_1(s) \geq \bar{s}_1^j\right\} \cap \mathcal{J}$.

4.4 Dynamic programming with cuts

In practice, it is unlikely that a decision-maker is interested in the whole set $\mathcal{Y}_N$ of (2o$_{pn}$.1c), which can be very large even for a small number of items. However, the decision-maker may want to define a range or *region of interest*.

Based on the respective application background, different scenarios may be considered:

(A) A minimal and maximal value for the second objective may be specified by the decision-maker, e. g., based on some practical constraints.

(B) A region of interest may be defined based on the selection of two supported points $\bar{s}_1^{j_1}, \bar{s}_1^{j_2}$ as $[\bar{s}_1^{j_1}, \bar{s}_1^{j_2}] \times [\bar{s}_2^{j_2}, \bar{s}_2^{j_1}]$, for $j_1, j_2 \in \{1, \ldots, q\},\ j_1 < j_2$.

(C) A natural choice for $\bar{s}_1^{j_1}$ and $\bar{s}_1^{j_2}$ in (B) are $\bar{s}^-$ and $\bar{s}^+$, that satisfy $\bar{s}^- = \bar{s}^j$ such that $j = \max\{\hat{j} \in \{1, \ldots, q\} : \bar{s}_2^{\hat{j}} \geq -W^2\}$ and $\bar{s}^+ = \bar{s}^{j+1}$, i. e., $\min\{\hat{j} \in \{1, \ldots, q\} : \bar{s}_2^{\hat{j}} < -W^2\} = j + 1$. In other words, $\bar{s}^-$ and $\bar{s}^+$ define the search triangle that contains the optimal point for the associated bi-dimensional KP.

Similarly, it is possible to define a region of interest by specifying two bounds, ε_1 for the first objective, and ε_2 for the second objective function. So $f_1(x)$ has to reach a lower bound ε_1, while $f_2(x)$ should not fall below a lower bound ε_2.

This could be used for the DP algorithm in the following way:

- Overall *DP*: The computation of new stages includes a new condition:

  $$S_k = \text{Dom}(S_{k-1} \cup \{(s_1 + p_k^1, s_2 + p_k^2, s_3 + w_k^1) \,:\, s_2 + p_k^2 \geq \varepsilon_2, s_3 + w_k^1 \leq W^1, s \in S_{k-1}\}).$$

- *DomR2*: States s in S_k could be discarded if $u_1(s) < \varepsilon_1$.

- *Bound*: (4.1) does not need to be checked for intervals with $\bar{s}_1^{j+1} < \varepsilon_1$ or $\bar{s}_2^{j} < \varepsilon_2$, $j \in \{1, \ldots, q\}$.

Then, the last stage S_n includes all nondominated points of the specified region of interest.

4.5 Computational results

The experiments were performed on an Intel Quadcore 2,80 GHz with 4 GB RAM. The implementation of the DP algorithm was coded in C++. It is based on a DP algorithm for (2o.1c) that was done by Marco Simões [see Figueira et al., 2013] and uses the MINKNAP algorithm of Pisinger [1997]. To compare with the results of a classical bi-dimensional approach, we used the cbc-solver from the Coin-OR-library [Forrest and Ralphs, 2015].

4.5.1 Experimental setup

We tested knapsack instances with 100 and 200 items. The instances of (2o$_{pn}$.1c) were generated according to the following types of correlation structures, with parameter $M = 10n$ and $\sigma = (M - 1)/30$:

Type A The profits p_i^1 and weights w_i^2 and w_i^1 are integers uniformly generated in the range $[1, M]$, i. e., profits $p_i^2 \in [-M, -1]$, for all $i = 1, \ldots, n$.

Type B The profits p_i^1 are integers uniformly generated in the range $[100, M-100]$, the weights w_i^2 and w_i^1 are normal distributed with expectation $\mu = p_i^1$ and standard deviation σ restricted to the range $[1, M-1]$. This induces a positive correlation between profits and weights, i. e., a negative correlation between profits p_i^1 and p_i^2 for (2o$_{pn}$.1c).

Type C The profits p_i^1 are integers uniformly generated in the range $[100, M-100]$, the weights w_i^2 and w_i^1 are normal distributed with expectation $\mu = p_i^1$ and $\mu = M - p_i^1$, respectively, and standard deviation σ restricted to the range $[1, M-1]$. This induces a positive correlation between profits p_i^1 and weights w_i^2, i. e., a negative correlation between profits p_i^1 and p_i^2 for (2o$_{pn}$.1c), and a negative correlation between profits p_i^1 and weights w_i^1.

Type D The profits p_i^1 are integers uniformly generated in the range $[100, M-100]$, the weights w_i^2 and w_i^1 are normal distributed with expectation $\mu = M - p_i^1$ and $\mu = p_i^1$, respectively, and standard deviation σ restricted to the range $[1, M-1]$. This induces a negative correlation between profits p_i^1 and weights w_i^2, i. e., a positive correlation between profits p_i^1 and p_i^2 for (2o$_{pn}$.1c), and a positive correlation between profits p_i^1 and weights w_i^1.

Type E The profits p_i^1 are integers uniformly generated in the range $[100, M-100]$, the weights w_i^2 and w_i^1 are normal distributed with expectation $\mu = M - p_i^1$ and standard deviation σ restricted to the range $[1, M-1]$. This induces a negative correlation between profits p_i^1 and weights w_i^2 and w_i^1, i. e., a positive correlation between profits p_i^1 and p_i^2 for (2o$_{pn}$.1c).

We remind, that the constraint slackness c_{W^1} is defined by $c_{W^1} \cdot \sum_{i=1}^{n} w_i^1 = W^1$ for a constraint $\sum_{i=1}^{n} w_i^1 \leq W^1$. Two values for the constraint slackness, namely $c_{W^2} = c_{W^1} = 0.25$ and $c_{W^2} = c_{W^1} = 0.75$, were applied for every type of instance. The complexity of the DP depends on the slackness of the hard constraint: On the one hand, a small constraint slackness limits the depth and, therefore, the number of states in the DP network. On the other hand, a large constraint slackness admits a large number of states. Hence, we chose values for the constraint slackness of $c_{W^2} = c_{W^1} = 0.25$ and $c_{W^2} = c_{W^1} = 0.75$ to test easy and hard instances for the DP algorithm, respectively.

The bounding induced by search zones was applied starting after the first half of all stages (i. e., after 50 and 100 items have been considered, respectively). This is reasonable because preliminary tests showed that in early stages the bounds are not tight enough to discard states, since most of the variables are not yet set. All presented results are the average of the results for ten random instances of the same type.

We computed the complete nondominated set using the DP algorithm. However, the motivation of our approach is to provide trade-off information between the profit of a solution and its level of constraint satisfaction. As mentioned before, this does not generally require to compute the whole nondominated set. Therefore, we also considered regions of interest of different sizes to analyze the performance of the algorithm in this context. More precisely, in our numerical experiments, the lower and upper bounds ε_1 and ε_2 (see Section 4.4) were generated using the set of extreme supported points $\mathcal{Y}_{eN}$. The two supported points defining the search triangle that contains the optimal solution of (1o.2c) (see Figure 4.4) shall be indicated by $\bar{s}^-$ and $\bar{s}^+$, with $\bar{s}_2^- \geq -W^2 \geq \bar{s}_2^+$ (see again Section 4.4). The two lexicographic maxima are given by $\bar{s}^1$ and $\bar{s}^q$. A region of interest of size $R \in [0,1]$ is then defined by $\varepsilon_1 = \bar{s}_1^- - R \cdot (\bar{s}_1^- - \bar{s}_1^1)$ and $\varepsilon_2 = \bar{s}_2^+ + R \cdot (\bar{s}_2^+ - \bar{s}_2^q)$. The regions of interest with $R = 1$, $R = 0.3$ and $R = 0$ are visualized for an exemplary problem instance in Figure 4.4. In particular, if $R = 0$, all nondominated points in the search triangle defined by $\bar{s}^-$ and $\bar{s}^+$ are computed.

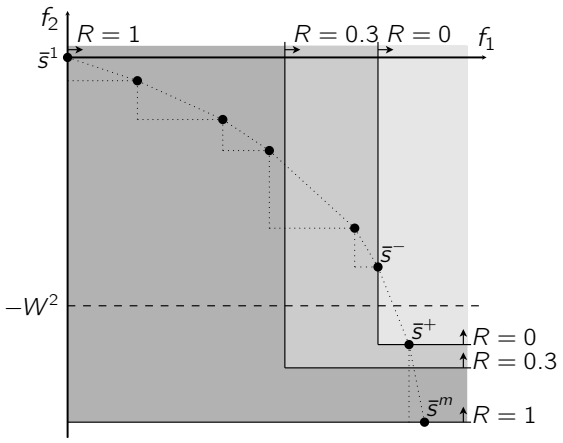


Figure 4.4: Illustration of regions of interest.

4.5.2 Computation of the nondominated set

In the classical (2o.1c) (with positive coefficients), every efficient solution is maximal in the sense that no further item can be included in the knapsack. As we know from Chapter 3, this is no longer true if negative coefficients occur. Therefore, a lot more combinations including partially filled knapsacks may lead to efficient solutions. Actually, the number of nondominated points of our instances (rows $|\mathcal{Y}_N|$ and column $R = 1$ in Tables 4.1 and 4.2) is considerably higher than in classical (2o.1c) [see Figueira et al., 2013]. As a consequence, the computational time for computing the whole set $\mathcal{Y}_N$ (rows DP and column $R = 1$ in Tables 4.1 and 4.2) is generally higher than for (2o.1c), while it is comparably fast with respect to the number of nondominated points. For instances with randomly chosen coefficients [Type A instances, analogous to Type A instances in Figueira et al., 2013] the CPU-time per computed solution is in both cases in the magnitude of

Type	R	0	0.25	0.50	0.75	1
A	DP	0.77	1.70	1.86	1.89	1.89
	cbc	3.26	83.37	105.23	115.00	117.74
	$\lvert\mathcal{Y}_N\rvert$	8.00	250.60	425.00	546.00	609.10
B	DP	8.84	16.90	18.35	19.04	19.04
	cbc	11.59	534.51	766.42	887.81	932.04
	$\lvert\mathcal{Y}_N\rvert$	9.22	957.30	1761.70	2397.90	2796.90
C	DP	8.70	15.10	18.05	18.86	19.08
	cbc	740.03	8884.81	20241.55	30183.00	33907.17
	$\lvert\mathcal{Y}_N\rvert$	208.89	2100.70	3862.50	5289.20	6174.10
D	DP	1.29	4.53	4.60	4.63	4.63
	cbc	28.05	1292.99	1319.41	1330.47	1333.19
	$\lvert\mathcal{Y}_N\rvert$	5.89	129.50	158.10	178.60	192.40
E	DP	0.01	0.02	0.03	0.03	0.03
	cbc	0.20	27.73	39.78	44.92	45.99
	$\lvert\mathcal{Y}_N\rvert$	1.30	68.50	110.30	155.90	195.70

Table 4.1: CPU-times of bi-objective and bi-dimensional approach in seconds and number of nondominated points for $n = 100$ items and $c_{W^2} = c_{W^1} = 0.25$.

Type	R	0	0.25	0.50	0.75	1
A	DP	0.10	2.27	4.00	4.32	4.31
	cbc	0.15	45.75	75.93	89.15	93.51
	$\lvert\mathcal{Y}_N\rvert$	2.667	608.60	1090.90	1401.40	1544.60
B	DP	10.69	60.16	90.62	105.15	107.83
	cbc	10.32	1173.51	2263.75	3054.50	3341.14
	$\lvert\mathcal{Y}_N\rvert$	10.44	3057.30	6385.80	8988.20	10680.60
C	DP	10.58	37.51	54.36	61.11	62.55
	cbc	106.11	1402.78	2756.58	3937.25	4345.56
	$\lvert\mathcal{Y}_N\rvert$	120.67	3552.50	7367.70	10659.60	12555.30
D	DP	0.24	14.18	15.99	16.16	16.39
	cbc	8.57	873.60	894.94	910.13	912.41
	$\lvert\mathcal{Y}_N\rvert$	2.44	215.20	279.80	346.10	389.80
E	DP	0.00	0.04	0.06	0.07	0.07
	cbc	0.23	95.77	140.92	156.31	159.43
	$\lvert\mathcal{Y}_N\rvert$	1.50	139.00	231.00	295.70	357.20

Table 4.2: CPU-times of bi-objective and bi-dimensional approach in seconds and number of nondominated points for $n = 100$ items and $c_{W^2} = c_{W^1} = 0.75$.

Type	R	0	0.01	0.02	0.05	0.10
A	DP	32.95	35.20	37.37	43.20	50.92
	cbc	48.76	101.26	152.66	313.95	540.98
	$\lvert\mathcal{Y}_N\rvert$	38.60	77.10	115.60	238.00	428.40
B	DP	292.92	364.54	402.57	439.44	462.87
	cbc	134.92	990.29	1738.43	3277.99	3982.18
	$\lvert\mathcal{Y}_N\rvert$	25.70	189.80	350.90	823.90	1539.20
C	DP	459.47	477.02	491.67	549.98	634.58
	cbc	19613.57	24399.59	63671.14	107250.20	248036,63
	$\lvert\mathcal{Y}_N\rvert$	348.80	680.70	1027.10	2067.30	3810.60
D	DP	34.43	94.71	138.78	180.81	183.94
	cbc	1956.22	17499.25	38604.51	158241.69	282236.19
	$\lvert\mathcal{Y}_N\rvert$	8.00	123.90	227.00	426.90	494.00
E	DP	0.03	0.06	0.08	0.13	0.20
	cbc	2.32	70.99	115.52	249.07	1008.64
	$\lvert\mathcal{Y}_N\rvert$	2.70	14.60	24.70	61.20	117.50

Table 4.3: CPU-times of bi-objective and bi-dimensional approach in seconds and number of nondominated points for $n = 200$ items and $c_{W^2} = c_{W^1} = 0.25$.

Type	R	0	0.01	0.02	0.05	0.10
A	DP	2.06	4.10	6.37	14.77	32.62
	cbc	1.20	25.64	50.43	91.67	154.21
	$\lvert\mathcal{Y}_N\rvert$	5.90	93.10	186.90	434.00	888.30
B	DP	309.35	562.24	648.94	810.26	1097.23
	cbc	70.29	1952.35	3140.26	4394.44	6343.19
	$\lvert\mathcal{Y}_N\rvert$	16.80	480.60	970.80	2396.80	4895.30
C	DP	519.12	562.00	602.17	732.26	953.67
	cbc	226.23	971.86	2046.62	4684.91	9679.14
	$\lvert\mathcal{Y}_N\rvert$	194.90	729.80	1277.20	2887.80	5589.50
D	DP	4.44	188.28	299.86	386.89	464.31
	cbc	14.71	26603.17	41680.23	45044.50	45270.46
	$\lvert\mathcal{Y}_N\rvert$	1.50	294.80	439.40	516.70	597.70
E	DP	0.02	0.11	0.18	0.34	0.54
	cbc	0.02	24.47	61.28	412.66	761.41
	$\lvert\mathcal{Y}_N\rvert$	1.10	51.50	101.00	225.10	390.90

Table 4.4: CPU-times of bi-objective and bi-dimensional approach in seconds and number of nondominated points for $n = 200$ items and $c_{W^2} = c_{W^1} = 0.75$.

milliseconds. All in all, the total computing times vary depending on the instance type, i. e., the correlation structure and the constraint slackness.

4.5.3 Regions of interest

The CPU-times for several sizes of regions of interest are listed in the rows DP of Tables 4.1 to 4.4. The instances with 200 items have a large CPU-time, hence, only small regions of interest were tested. For both problem sizes ($n = 100$ and $n = 200$), similar characteristics can be observed.

To illustrate the relation between the computation time and the number of computed nondominated points, these values are plotted for different values of R, using the case $R = 1$ as a reference (100%), in the plots of Figures 4.5 and 4.6 (for instances with 100 items). The symbol + always indicates the CPU-times, and the symbol $\circ$ represents the number of nondominated points.

In the case of $c_{W^2} = c_{W^1} = 0.25$, it can be seen that for small values of R a small amount of time is needed, but also the gained amount of information is small. With increasing values of R the required CPU-time grows very fast up to 100%. This means that for determining the nondominated points in a medium sized region of interest nearly all nondominated points have to be determined.

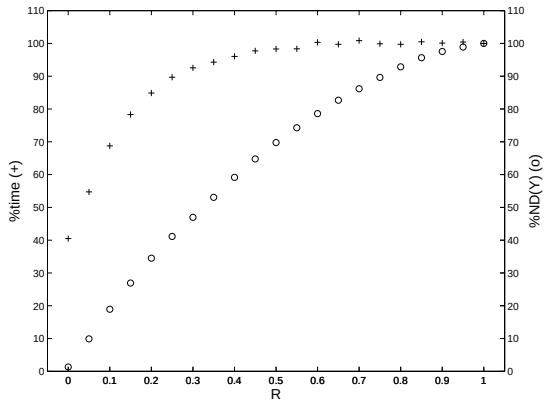
In the case of $c_{W^2} = c_{W^1} = 0.75$, the CPU-time grows at a smaller rate. As a consequence, the computing time corresponds approximately to the information gained by the computed nondominated points. A possible explanation could be that due to the larger number of solutions, the bounds ε_1 and ε_2 are stronger and the DP algorithm with cuts builds a smaller DP network.

The two graphs for problems of Type D have an interesting shape because the number of nondominated points is very large even for small regions of interest. In Figure 4.7 the nondominated points in the objective space are plotted for one exemplary instance. The correlation structure of Type D instances (positive correlation between both objective functions and between first objective function and constraint) induces a distribution with a flat angle between the points in the upper part of the nondominated set, and after a knee the slope gets very steep in the lower part. The constraint $\sum_{i=1}^{n} p_i^2 x_i \geq -W^2$ cuts the graph in the lower part, so for small values of R the region of interest includes already a large percentage of nondominated points.

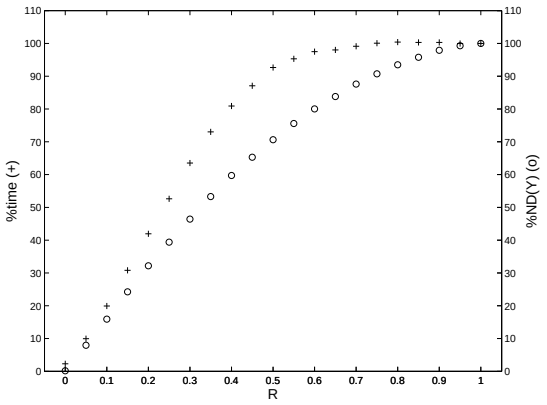
The CPU-times for instances of Type E are not strictly increasing for increasing values of R. However, in this case the computing times are very small and the deviations are in a magnitude of milliseconds.

4.5.4 Comparison of bi-dimensional knapsack problems and bi-objective knapsack problems with one positive and one negative objective function

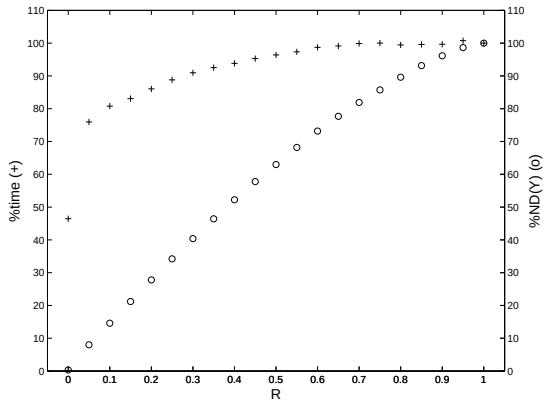
We use the cbc-solver from the Coin-OR-library to compare the DP based solution approach to the direct solution of (1o.2c). To get a fair comparison also in the case of search



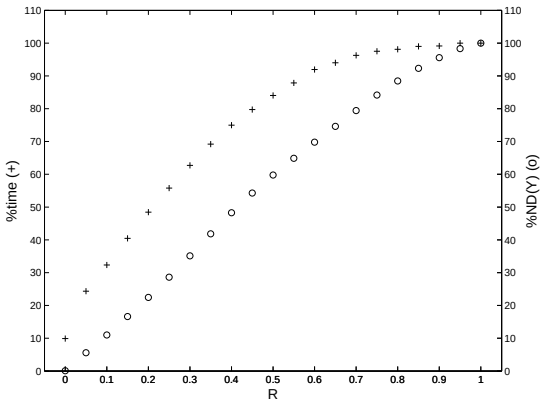
(a) Type A, $c_{W^2} = c_{W^1} = 0.25$.



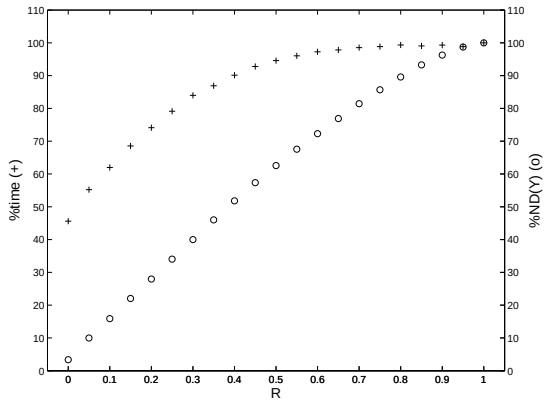
(b) Type A, $c_{W^2} = c_{W^1} = 0.75$.



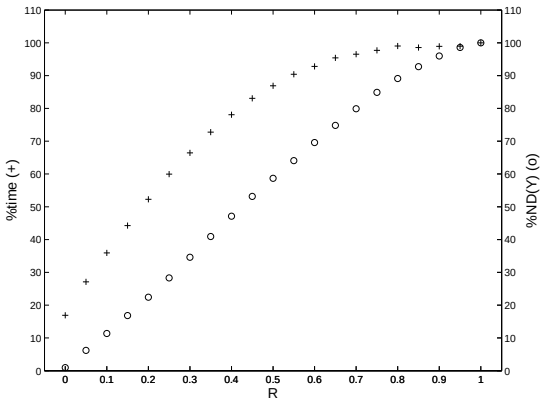
(c) Type B, $c_{W^2} = c_{W^1} = 0.25$.



(d) Type B, $c_{W^2} = c_{W^1} = 0.75$.

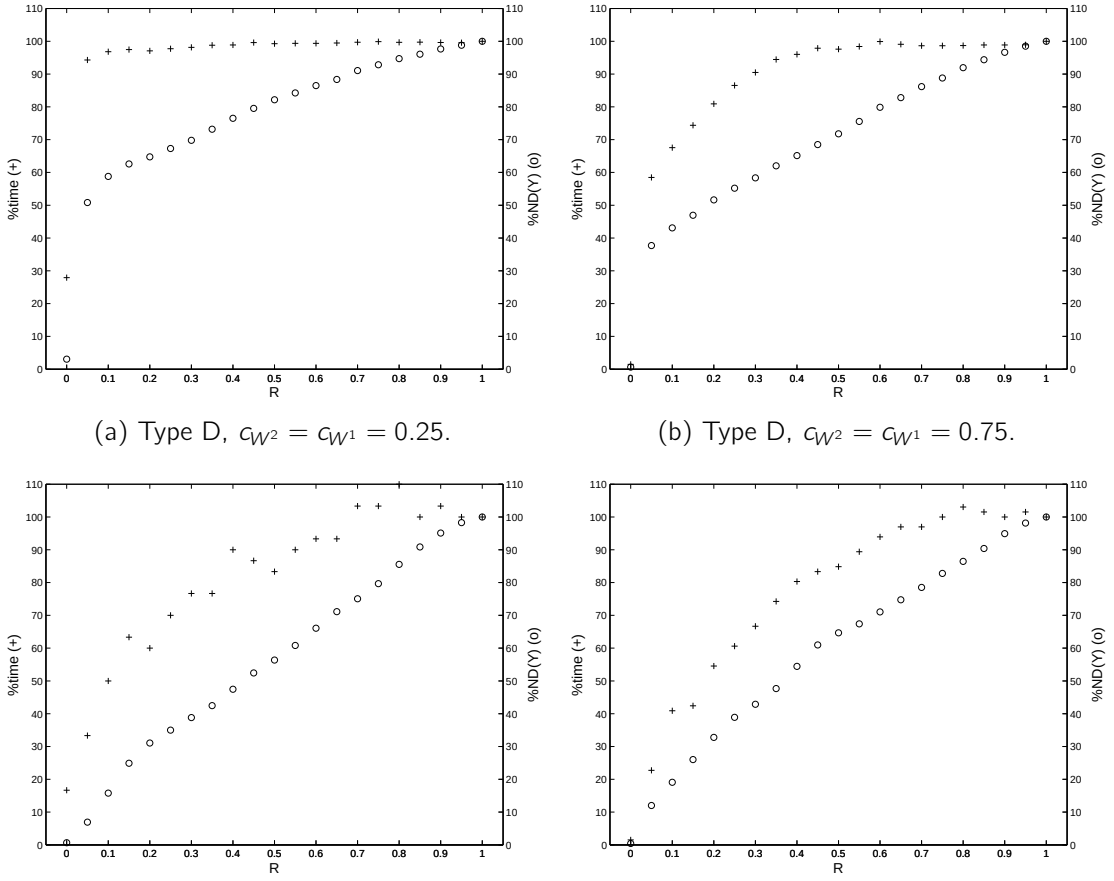


(e) Type C, $c_{W^2} = c_{W^1} = 0.25$.



(f) Type C, $c_{W^2} = c_{W^1} = 0.75$.

Figure 4.5: Computing times and number of nondominated points plotted for different values of R for instances of Types A, B and C ($n = 100$). The case $R = 1$ is used as a reference (100%).



(a) Type D, $c_{W^2} = c_{W^1} = 0.25$.

(b) Type D, $c_{W^2} = c_{W^1} = 0.75$.

(c) Type E, $c_{W^2} = c_{W^1} = 0.25$.

(d) Type E, $c_{W^2} = c_{W^1} = 0.75$.

Figure 4.6: Computing times and number of nondominated points plotted for different values of R for instances of Types D and E ($n = 100$). The case $R = 1$ is used as a reference (100%).

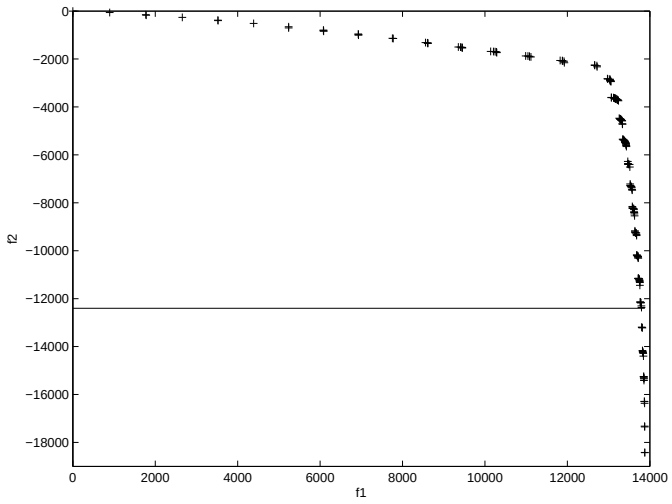


Figure 4.7: Set of nondominated points for one instance of Type D.

triangles, the time that would be needed by the bi-dimensional approach to compute all nondominated points was measured, i. e., the time needed to provide the same amount of information as the bi-objective approach. This can be realized by varying the capacity W^2 which, in turn, relates to applying the ε-constraint method (see Section 2.1) to the bi-objective problem. First, we set $W^2 = -\varepsilon_2$ and solve the associated knapsack problem. Afterward, we use the previously computed optimal solution $\bar{x}$ and set $W^2 = \sum_{i=1}^{n} w_i^2 \bar{x}_i - 1$ for the next instance. This continues until $f_1(\bar{x}) \leq \varepsilon_1$. The resulting CPU-times are presented in Tables 4.1 to 4.4 in the row cbc. One can observe, that the DP approach is always faster than solving all relevant bi-dimensional problems for regions of interest with $R = 0.01$ or greater.

Table 4.5 presents the average solution time for computing one solution of the bi-dimensional problem (1o.2c), where all nondominated points, i. e., $R = 1$, were computed for instances with 100 items and the nondominated points for $R = 0.1$ were taken into account for instances with 200 items. Especially for 200 items, one can observe a disagreement between CPU-times for solving one problem on average (Table 4.5) and computing all solutions for $R = 0$ (Tables 4.1 to 4.4). Solving one problem seems to be more expensive than solving several problems for the smallest region of interest. This results from a large variation in the solution times of the cbc-solver for varying right-hand side values; see standard deviations in Table 4.5. Some particular instances are very hard to solve for the solver and, hence, increase the average solution time over all runs, but not necessarily for small regions of interest. This behavior especially occurs for instances of Type D and of Type C for a slackness of 0.25 and is more extreme for instances with 200 items. In contrast, the DP-approach is robust against these changes since the constraint is considered as an objective function and all solutions are computed at once.

As expected, in most cases the bi-dimensional approach is faster in computing one specific solution, e. g., the optimal bi-dimensional solution. Surprisingly, instances with a

		$c_{W^2} = c_{W^1} = 0.25$		$c_{W^2} = c_{W^1} = 0.75$	
Type	n	100	200	100	200
A	t	0.19	1.23	0.06	0.17
	σ	0.19	0.49	0.04	0.10
B	t	0.33	2.54	0.31	1.31
	σ	0.40	2.21	0.25	1.46
C	t	5.46	65.12	0.35	2.20
	σ	12.91	138.30	0.50	56.87
D	t	6.58	530.79	2.27	72.56
	σ	16.72	3 399.41	5.37	181.80
E	t	0.24	8.69	0.46	1.72
	σ	0.55	308.12	1.17	6.17

Table 4.5: CPU-times (t in seconds) and standard deviation (σ) of cbc-solver solving (1o.2c) with different values of right-hand side (all nondominated points for $R = 1$ for instances with 100 items and for $R = 0.1$ for instances with 200 items).

negative correlation between original objective function and the remaining constraint, i. e., instances of Types D and E, seem to have a special structure, which makes it very efficient to apply the bi-objective approach. Furthermore, the bi-objective approach becomes dominant as soon as several solutions are requested. The DP algorithm is faster than the cbc-solver for most considered regions of interest.

4.6 Multi-dimensional knapsack problems with soft constraints

The idea of converting soft constraints into objective functions is directly applicable to knapsack problems with three or more constraints and with more than one objective function. When q of the d constraints of (mo.dc) are relaxed and transformed into objective functions, an associated instance of problem ((m+q)o.(d-q)c) is obtained, that has two types of objective functions, the first block of m objectives has only positive coefficients whereas the second block of q objectives has only negative coefficients, and $d-q$ remaining hard constraints.

As indicated in Section 2.2, the dynamic programming algorithm is still applicable to this type of problem. Including several objective functions and several constraints, this DP will suffer from the curse of dimensionality and can thus not be expected to be efficient but the optimal solution of the original problem can be computed as well as all nondominated points or a subset of nondominated points in a predefined region of interest. Due to the large number of states that can be expected in the dynamic programming algorithm it is very important to define an accurate region of interest.

In the case of one remaining hard constraint, a precomputation of supported points can

be applied using the results of Chapter 3. More research has to be done in this field to handle several constraints. The supported points can be used to determine the search zones for unsupported nondominated points [see Dächert et al., 2016, Klamroth et al., 2015, Dächert, 2014]. An accurate definition of a region of interest can be realized based on this information. Furthermore, we think that, as in the bi-objective case, the search zones are an important tool to introduce pruning strategies for the dynamic programming algorithm.

We conclude that the sensitivity analysis for (mo.dc) including soft or uncertain constraints provides an interesting line of research. A first step would be an extension of the approach to single-objective tri-dimensional knapsack problems including one or two soft constraints and to bi-objective bi-dimensional knapsack problems including one soft constraint.

4.7 Conclusion and further ideas

In this chapter we presented a bi-objective dynamic programming approach for solving the bi-dimensional knapsack problem with one soft constraint. The aim of this procedure is a sensitivity analysis on the right-hand side value of the soft or uncertain constraint to provide trade-off information. We applied a transformation, converting this constraint into an additional objective function. The resulting bi-objective knapsack problem has a special structure: The coefficients of the original objective function are all positive, the coefficients of the new objective function are all negative.

We applied a bi-objective dynamic programming algorithm which we adapted to this special structure. The extreme supported points are computed in a preprocessing step. Dominance relations are used to prune states of the DP network. Furthermore, we take advantage of the negative coefficients by introducing new bounds and search zones that are induced by the extreme supported points. If the bounds indicate that no extension of the currently analyzed state can be inside at least one search zone, all extensions of this state can be discarded.

We also presented a specialized algorithm with cuts, which enables the decision-maker to define a region of interest in which efficient solutions are determined. In this way, not the whole nondominated set is computed which, in general, considerably reduces computing times. This region of interest can be defined, for example, by specifying ranges of acceptable and/or interesting levels of constraint satisfaction.

The DP algorithm was tested on instances with 100 and 200 items, two different constraint slacknesses and five different correlation structures. We observed that, due to the structure of the bi-objective problem, the number of nondominated points is very large. Thus, the opportunity of defining a region of interest is usually of great importance for a decision-maker. The computational results indicate very good computing times in relation to the gained information.

We pointed out that the general idea of transforming soft constraints and applying a

dynamic programming algorithm is also applicable to higher dimensions. This approach gives rise to many aspect for future research.

An interesting question is whether some variables can be fixed during an extended pre-processing step. Especially in the case of small regions of interest, this seems to be a promising approach. A reduced set of variables implies less stages for the DP algorithm which would further reduce computing times.

5 Rectangular knapsack problems and representative solutions

In contrast to the previous chapters we present a single-objective single-dimensional knapsack problem in this chapter. We investigate a variant of the quadratic knapsack problem (Qo.1c), thus, the objective function is not a sum objective, but is defined by a quadratic term. For quadratic knapsack problems, in contrast to classical knapsack problems, the profit of a selection of items is not only determined by the individual profits, but also by profits generated by pairwise combinations of items. This can be used to model the fact that two items may complement each other such that their profit is increased if both of them are selected. The model still allows to model that two items are substitutes for each other by setting the combined profit equal to 0. In this case, including both items does not increase the profit over the sum of the individual profits.

The formulation of (Qo.1c) is very general and, therefore, its range of application is quite wide. For example, Johnson et al. [1993] present a problem in the context of compiler construction that may be formulated as a quadratic knapsack problem. Moreover, (Qo.1c) have been discussed in the context of the location of airports, freight handling terminals, railway stations, and satellite stations [Rhys, 1970, Witzgall, 1975].

(Qo.1c) is $\mathcal{NP}$-hard in the strong sense. However, we present a variant of (Qo.1c) which we call the *cardinality constrained rectangular knapsack problem* (Ro.Cc). The profit matrix is built by the product of two vectors and the constraint is a cardinality constraint.

(Ro.Cc) arises when solving a different problem. We consider the cardinality constrained bi-objective knapsack problem (2o.Cc)

$$\begin{aligned} \text{vmax} \quad & \left(\sum_{i=1}^{n} a_i x_i, \sum_{i=1}^{n} b_i x_i \right) \\ \text{s.t.} \quad & \sum_{i=1}^{n} x_i \leq k \\ & x_i \in \{0,1\}, \qquad i = 1, \ldots, n \end{aligned} \tag{2o.Cc}$$

where $a, b \in \mathbb{N}^n$, with $a, b \neq \mathbf{0}_n$, and $k \in \mathbb{N}$, $k < n$. Instead of computing the set of efficient solutions for this optimization problem, we want to find one (or several) representative nondominated point(s). Zitzler and Thiele [1998] introduced the *hypervolume indicator*, which is a quality measure for a representation based on the volume of the

objective space that is covered by the representative points. The authors presented this measure in the context of evolutionary algorithms and it is mainly used in this field. A reference point has to be defined to compute the hypervolume that is spanned by the representative set and the reference point. Choices for the reference point are, for example the origin or, if available, the nadir point of the given problem. The problem of finding one solution of (2o.Cc) that maximizes the hypervolume, considering $(0,0)^\top$ as reference point, is equivalent to (Ro.Cc).

The structure of (Ro.Cc) allows to formulate a polynomial time 4.5-approximation algorithm, where, for some $\rho > 1$, an algorithm is called a *polynomial time ρ-approximation algorithm*, if it computes a feasible solution in run time being polynomial in the coding length of the input such that

$$\rho \geq \max\left\{\frac{\mathrm{OPT}}{\mathrm{ALG}}, \frac{\mathrm{ALG}}{\mathrm{OPT}}\right\}.$$

Here, OPT denotes the optimal objective function value of the maximization problem and ALG the objective function value of the solution which is the output of the algorithm [Cormen et al., 2001].

The remainder of this chapter is organized as follows: In Section 5.1 we give an introduction to quadratic knapsack problems. We introduce the cardinality constrained rectangular knapsack problem in Section 5.2 and present upper and lower bounds. These bounds motivate an approximation algorithm that is formulated in Section 5.3. Furthermore, an approximation ratio ρ is proven for this algorithm. We also introduce an improved version of the approximation algorithm. In Section 5.4 we present a computational study on both algorithms. The application of the cardinality constrained rectangular knapsack problem for computing a representative solution of (2o.Cc) is discussed in Section 5.5. Further ideas for computing a representative set of solutions are outlined. Section 5.6 provides a conclusion and further ideas.

5.1 Quadratic knapsack problems

Gallo et al. [1980] first introduced the *binary quadratic knapsack problem* (Qo.1c). It is an expansion of the classical knapsack problem and can be concisely stated as follows: Given n items, the profit for including item i is given by the coefficient p_{ii}. Additionally, a profit $p_{ij} + p_{ji}$ is generated if both items i and j are selected. All profits p_{ij} are assumed to be non-negative integer values and can be compactly written in a profit matrix

$$P := (p_{ij})_{\substack{i=1,\dots,n \\ j=1,\dots,n}}.$$

The profits p_{ij} and p_{ji} are either both realized, i. e., if items i and j are selected, or both not realized, i. e., if item i or item j is not selected. Hence, p_{ij} and p_{ji} can be assumed to be equally valued, which results in a symmetric matrix P. However, for the rectangular

knapsack problem we assume another structure for the coefficients of P that we introduce in the next section.

As for (1o.1c), each item i has a positive integral weight w_i and the goal is to select a subset of items that maximizes the overall profit and whose sum of weights does not exceed the given knapsack capacity W. All assumptions on the constraint are the same as for the classical knapsack problem, cf. Section 2.2. Additionally, a simple problem reduction is possible if $w_i + w_j > W$. This case indicates that items i and j cannot be selected together, hence, the combined profit $p_{ij} + p_{ji}$ will never be realized and can be set to 0. As usual, the binary decision variable x_i indicates if item i is selected, $x_i = 1$, or not, $x_i = 0$. Thus, (Qo.1c) can be defined as follows:

$$\begin{aligned} \max \quad & x^\top P x = \sum_{i=1}^{n} \sum_{j=1}^{n} p_{ij} x_i x_j \\ \text{s.t.} \quad & \sum_{i=1}^{n} w_i x_i \leq W \\ & x_i \in \{0,1\}, \qquad i = 1, \ldots, n. \end{aligned} \tag{Qo.1c}$$

An illustrative interpretation of (Qo.1c) can be given based on graphs. We define a complete undirected graph $G = (V, E)$ with vertex set $V = \{1, \ldots, n\}$, i. e., each vertex corresponds to one item of (Qo.1c). Each vertex has assigned a profit value p_{ii} and a weight value w_i and each edge (i, j) has assigned a profit value $p_{ij} + p_{ji}$.

Definition 5.1 *Given a graph $G = (V, E)$, a* clique *is defined as a subset of vertices $V' \subset V$ where every pair of vertices $i, j \in V'$ is connected by an edge $(i, j) \in E$.*

A selected clique $S \subset V$ is called feasible for (Qo.1c) if the overall assigned weight does not exceed the capacity W. The overall profit of S consists of the profit assigned to the vertices and edges assigned to S, i. e., an edge (i, j) represents the profit that is generated since both vertices i and j are elements of the clique S. The optimization problem can now be formulated as selecting a feasible clique $S \subset V$ that realizes the maximal profit.

It is well known that the quadratic knapsack problem is $\mathcal{NP}$-complete in the strong sense, which can be shown by a polynomial reduction to the *Clique*-problem [Garey and Johnson, 1979, Pisinger, 2007]:

Definition 5.2 *Given a graph $G = (V, E)$ and a positive integer $k \leq |V|$, the* Clique-*problem asks, if G contains a clique of size k or more.*

A corresponding instance of (Qo.1c) can be modeled by setting the coefficients as follows: $n := |V|$; $p_{ii} := 0$, for $i = 1, \ldots, n$; $p_{ij}, p_{ji} := 1$ if $(i, j) \in E$ and $p_{ij}, p_{ji} := 0$ otherwise, for $i \neq j$, $i, j = 1, \ldots, n$; $w_i := 1$ for $i = 1, \ldots, n$; and $W := k$. Thus, only combinations of items/vertices that are directly connected contribute to the objective function value. The constraint of this special instance reduces to a cardinality constraint, $\sum_{i=1}^{n} x_i \leq k$, that is completely exploited by the optimal solution. Hence, if and only if

the optimal objective function value is equal to $k(k-1)$, the answer to the clique problem is positive.

On the one hand, the quadratic knapsack problem has been widely studied in the literature, see Pisinger [2007] for a comprehensive survey. Exact solution algorithms are mainly based on branch-and-bound (BB) schemes. Besides the model of (Qo.1c), Gallo et al. [1980] also presented the first BB algorithm for this optimization problem. Their approach makes use of upper bounds that are computed by a relaxed version of (Qo.1c). The objective function is replaced by an upper plane, that is a linear function g such that $g(x) \geq x^\top Px$ for any feasible solution x of (Qo.1c). Solving this new optimization problem, which is a (1o.1c) due to the linear objective function, yields an upper bound on (Qo.1c).

Caprara et al. [1999] also use upper planes for computing upper bounds. In addition, they present a reformulation to an equivalent instance of (Qo.1c) using Lagrangian relaxation. The reformulated instance provides a tight upper bound at the root node of the BB algorithm. Billionnet et al. [1999] use upper bounds based on Lagrangian decomposition and, in a more recent version, Billionnet and Soutif [2004] introduce algorithms for fixing variables based on these bounds. These algorithms are used to reduce the size of the problem before applying the BB. Pisinger et al. [2007] introduce an aggressive reduction algorithm for large instances of (Qo.1c), where *aggressive* means that a large effort is spent for the reduction such that the final optimization is done rather easy. The authors use the bounds of Caprara et al. [1999] and Billionnet et al. [1999] for the reduction of (Qo.1c). Rodrigues et al. [2012] present a linearization scheme for (Qo.1c) that provides tight upper bounds for a BB algorithm.

On the other hand, rather few results are known about the approximation of (Qo.1c). Since the problem is strongly $\mathcal{NP}$-hard, a fully polynomial approximation scheme (FPTAS) cannot be expected unless $\mathcal{P} = \mathcal{NP}$. Furthermore, it is unknown whether there exists an approximation with a constant approximation ratio for (Qo.1c). Taylor [2016] present an approximation algorithm based on an approach for the densest k-subgraph problem. They show that for $\varepsilon > 0$, (Qo.1c) can be approximated with an approximation ratio in $\mathcal{O}(n^{2/5+\varepsilon})$ and a run time of $\mathcal{O}(n^{9/\varepsilon})$. Rader and Woeginger [2002] prove that for a variant of (Qo.1c), where positive as well as negative profit coefficients p_{ij} are considered, there does not exist any polynomial time approximation algorithm with finite worst case guarantee unless $\mathcal{P} = \mathcal{NP}$.

Other approximation results concentrate on special cases of (Qo.1c) where the underlying graph $G = (V, E)$, with $E = \{(i, j) : i, j \in V, i \neq j, p_{ij} \neq 0\}$, has a specific structure. Pferschy and Schauer [2016] present an FPTAS for (Qo.1c) on graphs of bounded treewidth, which includes series-parallel graphs [see Bodlaender and Koster, 2008]. Furthermore, the authors introduce a polynomial time approximation scheme (PTAS) for graphs that do not contain any fixed graph H as a minor, which includes planar graphs. As negative results, Pferschy and Schauer [2016] show that (Qo.1c) on 3-book embeddable graphs is strongly $\mathcal{NP}$-hard and Rader and Woeginger [2002] prove that (Qo.1c) on vertex series-parallel graphs is strongly $\mathcal{NP}$-hard.

Kellerer and Strusevich [2010] introduce a very special variant of (Qo.1c): the symmetric quadratic knapsack problem. In addition to assigning a profit to pairs of items that both have been selected this variant also assigns a profit to pairs of items that both have not been selected. Furthermore, the profits p_{ij} are built as a multiplicative of two coefficients of which one also defines the weight of the constraint. Kellerer and Strusevich [2010] introduce an FPTAS to solve the problem, which is further improved by Xu [2012].

5.2 Cardinality constrained rectangular knapsack problems

The cardinality constrained rectangular knapsack problem (Ro.Cc) is a variant of (Qo.1c) with the following properties:

$$\begin{aligned} \max \quad & f(x) = x^\top a\, b^\top x = \sum_{i=1}^{n}\sum_{j=1}^{n} a_i b_j\, x_i x_j \\ \text{s.t.} \quad & \sum_{i=1}^{n} x_i \leq k \\ & x_i \in \{0,1\}, \qquad i = 1,\ldots,n \end{aligned} \tag{Ro.Cc}$$

where $a, b \in \mathbb{N}^n$, with $a, b \neq \mathbf{0}_n$, and $k \in \mathbb{N}$, $k < n$. Note, that $P = a\, b^\top$, i. e., $\operatorname{rank}(P) = 1$, with $p_{ij} = a_i b_j$ and $p_{ji} = a_j b_i$, i. e., in general, P is not symmetric. We assume that $k \geq 2$. Otherwise, i. e., if $k = 1$, the problem reduces to finding the largest coefficient $a_i b_i$, for $i \in \{1,\ldots,n\}$.

The rectangular objective function is formulated in analogy to the Koopmans-Beckmann form of the quadratic assignment problem, see Burkard et al. [1998], which is also a particular case of the more general Lawler formulation. In both cases, the two respective four dimensional arrays of profit/cost coefficients are given as a product of lower dimensional parameters.

5.2.1 Illustrative interpretation

The denotation *rectangular* knapsack problem is motivated by the special structure of P given by the coefficients $a_i b_j$. Every coefficient can be interpreted as the area of a rectangle. Accordingly, for fixed item $\hat{\imath} \in \{1,\ldots,n\}$ all rectangles corresponding to coefficients $a_{\hat{\imath}} b_j$, $j = 1,\ldots,n$, have the same width, and all rectangles corresponding to coefficients $a_j b_{\hat{\imath}}$, $j = 1,\ldots,n$, have the same height. Note that, the objective function can be rewritten as

$$f(x) = x^\top a\, b^\top x = (a^\top x) \cdot (b^\top x) = \sum_{i=1}^{n} a_i x_i \cdot \sum_{i=1}^{n} b_i x_i,$$

which can be interpreted as choosing a subset $S \subset \{1,\ldots,n\}$ of items such that the area of the rectangle with width $\sum_{i\in S} a_i$ and height $\sum_{i\in S} b_i$ is maximized.

Example 5.3 *We consider the following instance of* (Ro.Cc)*:*

$$\begin{aligned} \max \quad & \left((4,5,2,12,7)^\top x\right) \cdot \left((6,3,8,5,10)^\top x\right) \\ \text{s.t.} \quad & \sum_{i=1}^{5} x_i \leq 2 \\ & x_i \in \{0,1\}, \qquad i = 1,\ldots,5 \end{aligned}$$

The corresponding rectangles are plotted in Figure 5.1. Each rectangle has the same position in the overall rectangle as the corresponding coefficient $p_{ij} = a_i b_j$ in the profit matrix P. The optimal solution $x = (0,1,0,0,1)^\top$ generates an objective function value that corresponds to the highlighted area in the figure.

	b_1	b_2	b_3	b_4	b_5
a_1	4 · 6	4 · 3	4 · 8	4 · 5	4 · 10
a_2	5 · 6	5 · 3	5 · 8	5 · 5	5 · 10
a_3	2 · 6	2 · 3	2 · 8	2 · 5	2 · 10
a_4	12 · 6	12 · 3	12 · 8	12 · 5	12 · 10
a_5	7 · 6	7 · 3	7 · 8	7 · 5	7 · 10

Figure 5.1: Visualization of coefficients $p_{ij} = a_i b_j$, interpreted as areas of rectangles.

5.2.2 Bounds

The setup of the profit matrix P implies an easy computation of bounds for (Ro.Cc). In the following we assume that all instances are defined or reordered such that

$$a_1 \geq \ldots \geq a_n$$

and in case of ties, i. e., if $a_i = a_{i+1}$ for $i \in \{1,\ldots,n-1\}$, such that

$$b_i \geq b_{i+1}.$$

Let $\mathcal{S}_n$ denote the symmetric group of order n and $\pi \in \mathcal{S}_n$ denote a permutation of $\{1,\ldots,n\}$. More specifically, consider π such that

$$b_{\pi(1)} \geq \ldots \geq b_{\pi(n)}$$

and in case of ties, i. e., if $b_{\pi(j)} = b_{\pi(j+1)}$ for $j \in \{1, \dots, n-1\}$, such that

$$a_{\pi(j)} \geq a_{\pi(j+1)}.$$

Using the sorted coefficients a_i, $b_{\pi(j)}$ of the objective, one can compute an upper bound for (Ro.Cc) in a straight forward way.

Lemma 5.4 *For every feasible solution $x \in \{0,1\}^n$ of* (Ro.Cc) *the following inequality holds:*

$$f(x) \leq \sum_{i=1}^{k} a_i \cdot \sum_{j=1}^{k} b_{\pi(j)} = \mathcal{U}$$

This bound is tight, if

$$\{\pi(j) : 1 \leq j \leq k\} = \{1, \dots, k\}. \tag{5.1}$$

Note that, in general, this upper bound does not correspond to a solution of (Ro.Cc) since the value of a variable x_i may be differently defined w. r. t. the respective sorting of the coefficients. As soon as Equation 5.1 holds, the upper bound $\mathcal{U}$ corresponds to a solution of (Ro.Cc) and this solution is optimal.

Proof. We consider the objective function of (Ro.Cc):

$$f(x) = \sum_{i=1}^{n} a_i x_i \cdot \sum_{i=1}^{n} b_i x_i = \sum_{i=1}^{n} a_i x_i \cdot \sum_{j=1}^{n} b_{\pi(j)} x_{\pi(j)}.$$

The cardinality constraint restricts the number of selected items to k. Due to the ordering of coefficients a_i we know that

$$0 \leq \sum_{i=1}^{n} a_i x_i \leq \sum_{i=1}^{k} a_i$$

for every feasible solution x of (Ro.Cc). Analogously, due to the definition of the permutation π we know that

$$0 \leq \sum_{j=1}^{n} b_{\pi(j)} x_{\pi(j)} \leq \sum_{j=1}^{k} b_{\pi(j)}$$

for every feasible solution x of (Ro.Cc). Thus,

$$f(x) = \sum_{i=1}^{n} a_i x_i \cdot \sum_{j=1}^{n} b_{\pi(j)} x_{\pi(j)} \leq \sum_{i=1}^{k} a_i \cdot \sum_{j=1}^{k} b_{\pi(j)} = \mathcal{U}.$$

Furthermore, if $\{\pi(j) : 1 \leq j \leq k\} = \{1, \dots, k\}$, the upper bound is based on the selection of the k items $1, \dots, k$:

$$\sum_{i=1}^{k} a_i \cdot \sum_{j=1}^{k} b_{\pi(j)} = \sum_{i=1}^{k} a_i \cdot \sum_{i=1}^{k} b_i = \sum_{i=1}^{n} a_i x_i \cdot \sum_{i=1}^{n} b_i x_i$$

with

$$x_i = \begin{cases} 1 & \text{, for } i \in \{1, \dots, k\} \\ 0 & \text{, otherwise} \end{cases}$$

The solution x is feasible and realizes $\mathcal{U}$. Hence, x is optimal and $\mathcal{U}$ is a tight bound. □

A lower bound on (Ro.Cc) can be obtained by using the sorting of the coefficients again. Let $\tilde{x}$ and $\hat{x} \in \{0,1\}^n$ be defined as follows:

$$\tilde{x}_i = \begin{cases} 1 & \text{, for } i \in \left\{1, \dots, \left\lceil \frac{k}{2} \right\rceil\right\} \cup \left\{\pi(1), \dots, \pi\left(\left\lfloor \frac{k}{2} \right\rfloor\right)\right\} \\ 0 & \text{, otherwise} \end{cases} \tag{5.2}$$

$$\hat{x}_i = \begin{cases} 1 & \text{, for } i \in \left\{1, \dots, \left\lfloor \frac{k}{2} \right\rfloor\right\} \cup \left\{\pi(1), \dots, \pi\left(\left\lceil \frac{k}{2} \right\rceil\right)\right\} \\ 0 & \text{, otherwise} \end{cases} \tag{5.3}$$

For notational convenience, let $\kappa := \frac{k}{2}$, $\overline{\kappa} := \left\lceil \frac{k}{2} \right\rceil$, and $\underline{\kappa} := \left\lfloor \frac{k}{2} \right\rfloor$. If k is even, the equality $\overline{\kappa} = \underline{\kappa} = \frac{k}{2}$ holds, i. e., $\tilde{x}$ and $\hat{x}$ are identical.

Remark 5.5 *The definition of $\tilde{x}$ guarantees that at least the product*

$$\sum_{i=1}^{\overline{\kappa}} a_i \cdot \sum_{j=1}^{\underline{\kappa}} b_{\pi(j)}$$

is realized in the objective function. Due to the ordering of coefficients a_i and $b_{\pi(j)}$ this is the maximal possible value that a product of $\overline{\kappa}$ coefficients a_i and $\underline{\kappa}$ coefficients b_j can achieve. The same holds analogously for $\hat{x}$. This property is important to prove an approximation quality in the following, see the proof of Theorem 5.9.

Lemma 5.6 *For an optimal solution x^* of* (Ro.Cc) *the following inequality holds:*

$$f(x^*) \geq \max\left\{f(\tilde{x}), f(\hat{x})\right\} = \mathcal{L}.$$

Proof. The solutions $\tilde{x}$ and $\hat{x}$ are both elements of $\{0,1\}^n$. The sets $\{1, \dots, \overline{\kappa}\}$ and $\{\pi(1), \dots, \pi(\overline{\kappa})\}$ have cardinality $\overline{\kappa}$ and the sets $\{\pi(1), \dots, \pi(\underline{\kappa})\}$ and $\{1, \dots, \underline{\kappa}\}$ have cardinality $\underline{\kappa}$. Therefore, it holds:

$$\left.\begin{aligned} \sum_{i=1}^{n} \tilde{x}_i &\leq \overline{\kappa} + \underline{\kappa} \\ \sum_{i=1}^{n} \hat{x}_i &\leq \underline{\kappa} + \overline{\kappa} \end{aligned}\right\} = k.$$

Note that equality is obtained if the sets $\{1, \dots, \overline{\kappa}\}$ and $\{\pi(1), \dots, \pi(\underline{\kappa})\}$ ($\{1, \dots, \underline{\kappa}\}$ and $\{\pi(1), \dots, \pi(\overline{\kappa})\}$, respectively) are disjoint. If the sets are not disjoint, the bound can be improved by including more items. We discuss this in Section 5.3.1.

Both solutions $\tilde{x}$ and $\hat{x}$ are feasible for (Ro.Cc) and the corresponding objective function values are lower bounds on the optimum. □

We define $\widetilde{\mathcal{L}} := f(\tilde{x})$ and $\widehat{\mathcal{L}} := f(\hat{x})$. The following example draws a connection between the bound computation and the visualization of (Ro.Cc) as a selection of a subset of rectangular areas.

Example 5.7 *Consider the following instance of* (Ro.Cc)*:*

$$\begin{aligned} \max \quad & \big((6,5,5,4,3,3,2,1)^\top x\big) \cdot \big((6,11,4,10,6,9,1,8)^\top x\big) \\ \text{s.t.} \quad & \sum_{i=1}^{8} x_i \leq 5 \\ & x_i \in \{0,1\}, \qquad i = 1,\dots,8. \end{aligned}$$

Thus, the permutation π is $\pi = (2,4,6,8,1,5,3,7)^\top$ and the permuted vector b_π is given by $b_\pi = (11,10,9,8,6,6,4,1)^\top$.

As described above, the coefficients $a_i \cdot b_{\pi(j)}$, for $i,j = 1,\dots,8$, can be interpreted as rectangles with width a_i and height $b_{\pi(j)}$ and, consequently, with area $a_i \cdot b_{\pi(j)}$. We arrange the rectangles line by line according to the index i, and column by column according to the index $\pi(j)$ (see Figure 5.2). In doing so, the rectangles representing the coefficients are sorted in non-increasing manner from top to bottom and from left to right. Feasible solutions of (Ro.Cc) *correspond to 5^2 rectangles, which have to be part of intersections of rows and columns with equal sets of indices $\mathcal{I}$, i. e., a set of indices $\mathcal{I} \subset \{1,\dots,8\}$ with $|\mathcal{I}| \leq 5$.*

- *The upper bound computation chooses the 5 largest rows and columns, i. e., a_1 to a_5 and $b_{\pi(1)}$ to $b_{\pi(5)}$. In our example, we obtain:*

  $$\mathcal{U} = \sum_{i=1}^{5} a_i \cdot \sum_{j=1}^{5} b_{\pi(j)} = (6+5+5+4+3)\cdot(11+10+9+8+6) = 23 \cdot 44 = 1012.$$

  This corresponds to the area of the 5^2 largest rectangles in the upper left part of the overall rectangle in Figure 5.2.

- *For the lower bound computation at most 5 variables corresponding to the first three and two (two and three, respectively) indices of rows and columns are selected. In doing so, the largest $2 \cdot 3$ rectangles in the upper left part of the overall rectangle in Figure 5.3 (lower bound $\widehat{\mathcal{L}}$) are included in the solution and, in addition, feasibility is guaranteed. In the example, the candidate solutions are $\tilde{x} = (1,1,1,1,0,0,0,0)^\top$ and $\hat{x} = (1,1,0,1,0,1,0,0)^\top$. The lower bound is computed as:*

  $$\begin{aligned} \mathcal{L} &= \max\big\{\widetilde{\mathcal{L}}, \widehat{\mathcal{L}}\big\} \\ &= \max\big\{(6+5+5+4)\cdot(6+11+4+10), (6+5+4+3)\cdot(6+11+10+9)\big\} \\ &= \max\big\{620, 648\big\} = 648. \end{aligned}$$

The optimal solution of this instance is $x^* = (1,1,1,1,0,1,0,0)^\top$ with $f(x^*) = 920$. We can verify that indeed: $\mathcal{U} = 1012 \geq 920 \geq 648 = \mathcal{L}$.

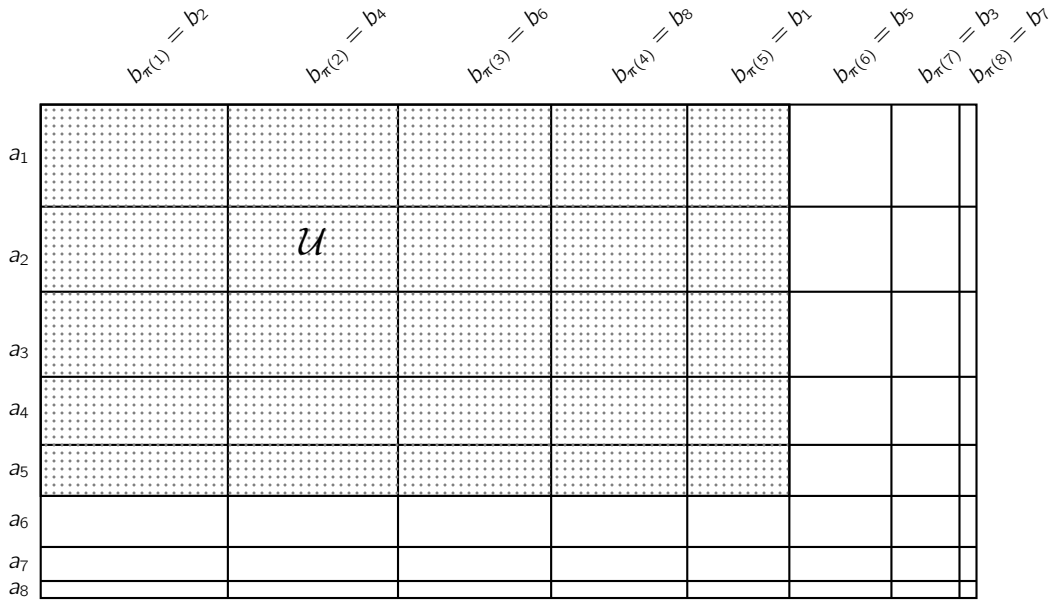


Figure 5.2: Area that defines the upper bound $\mathcal{U}$ (⊡) for Example 5.7.

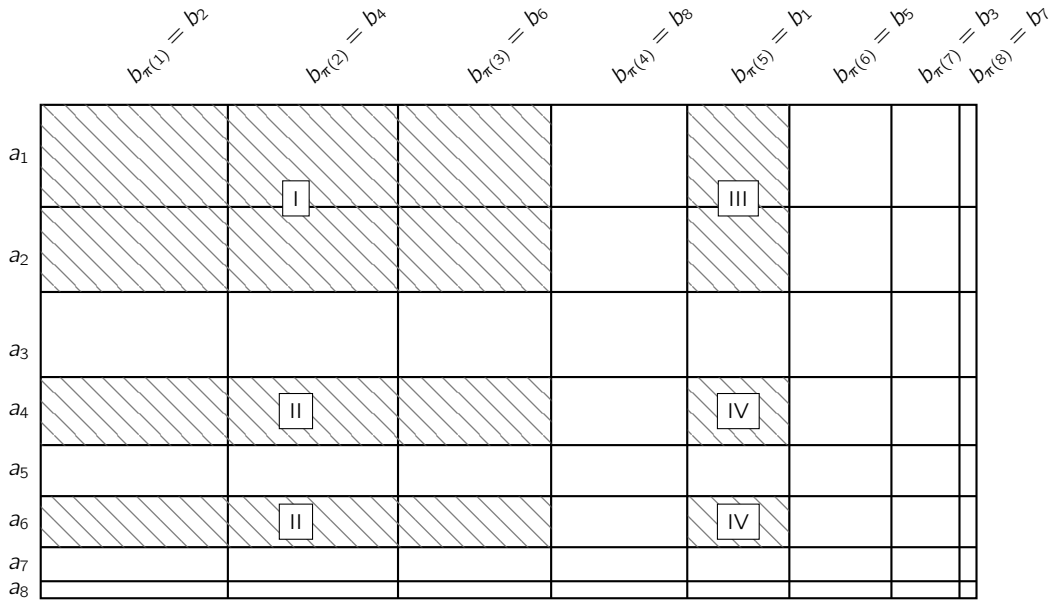


Figure 5.3: Area that defines the lower bound $\mathcal{L} = \widehat{\mathcal{L}}$ (▨) for Example 5.7. The assignment of labels I to IV become relevant for the proof of (5.4).

In this context, we show that the following inequality holds:

$$\mathcal{L} \geq \sum_{i=1}^{\underline{\kappa}} \sum_{j=1}^{\overline{\kappa}} a_i b_{\pi(j)}. \tag{5.4}$$

Referring to the description of Example 5.7, the right-hand side of this inequality corresponds to the area of the $\overline{\kappa} \cdot \underline{\kappa}$ largest rectangles in the left upper part of the overall rectangle (see also Remark 5.5). We partition the area corresponding to the lower bound into four distinct areas to show that the inequality holds. If we would apply the following derivation to Example 5.7, the four terms resulting from this subdivision correspond, in this order, to the four areas (I to IV) in Figure 5.3. Thus, the figure can be used to illustrate the following proof of inequality 5.4:

$$\begin{aligned}
\mathcal{L} \geq \widehat{\mathcal{L}} &= \sum_{i=1}^{n} \sum_{j=1}^{n} a_i b_j \, \hat{x}_i \hat{x}_j \\
&= \sum_{i=1}^{\underline{\kappa}} \sum_{j=1}^{\overline{\kappa}} a_i b_{\pi(j)} + \sum_{\substack{i=1 \\ \pi(i) \notin \{1,\ldots,\underline{\kappa}\}}}^{\overline{\kappa}} \sum_{j=1}^{\overline{\kappa}} a_{\pi(i)} b_{\pi(j)} \\
&\quad + \sum_{i=1}^{\underline{\kappa}} \sum_{\substack{j=1 \\ j \notin \{\pi(1),\ldots,\pi(\overline{\kappa})\}}}^{\underline{\kappa}} a_i b_j + \sum_{\substack{i=1 \\ \pi(i) \notin \{1,\ldots,\underline{\kappa})\}}}^{\overline{\kappa}} \sum_{\substack{j=1 \\ j \notin \{\pi(1),\ldots,\pi(\overline{\kappa})\}}}^{\underline{\kappa}} a_{\pi(i)} b_j \\
&\geq \sum_{i=1}^{\underline{\kappa}} \sum_{j=1}^{\overline{\kappa}} a_i b_{\pi(j)}
\end{aligned}$$

Analogously, using the definition of $\tilde{x}$, it holds that:

$$\mathcal{L} \geq \sum_{i=1}^{\overline{\kappa}} \sum_{j=1}^{\underline{\kappa}} a_i b_{\pi(j)}. \tag{5.5}$$

5.3 Approximation algorithms

The results of Section 5.2.2 do naturally motivate an approximation algorithm, see Algorithm 5.1. It computes the solutions $\tilde{x}$ and $\hat{x}$ and outputs the better alternative as approximate solution.

The computation of $\tilde{x}$ and $\hat{x}$ and of their objective function values $\widetilde{\mathcal{L}}$ and $\widehat{\mathcal{L}}$ can be realized in time $\mathcal{O}(n)$. Therefore, with a time complexity of $\mathcal{O}(n \log n)$, the sorting of the coefficients is the most time consuming part of Algorithm 5.1.

Lemma 5.8 *Algorithm 5.1 has a time complexity of* $\mathcal{O}(n \log n)$.

Algorithm 5.1 Approximation algorithm for (Ro.Cc)

Input: coefficients $a = (a_1, \ldots, a_n)^\top$, $b = (b_1, \ldots, b_n)^\top$, capacity k

$\tilde{x} := \mathbf{0}_n$, $\hat{x} := \mathbf{0}_n$, $\overline{\kappa} := \lceil \frac{k}{2} \rceil$ and $\underline{\kappa} := \lfloor \frac{k}{2} \rfloor$

compute permutation $\pi \in \mathcal{S}_n$ such that

$$\begin{cases} b_{\pi(j)} > b_{\pi(j+1)}, \text{ or} \\ b_{\pi(j)} = b_{\pi(j+1)} \text{ and } a_{\pi(j)} \geq a_{\pi(j+1)} \end{cases} \quad \text{for } j = 1, \ldots, n-1$$

for $i := 1, \ldots, \underline{\kappa}$ **do** *// set $\tilde{x}$ and $\hat{x}$ analogous to 5.2 and 5.3*

  $\tilde{x}_i := 1$, $\tilde{x}_{\pi(i)} := 1$

  $\hat{x}_i := 1$, $\hat{x}_{\pi(i)} := 1$

$\tilde{x}_{\overline{\kappa}} := 1$

$\hat{x}_{\pi(\overline{\kappa})} := 1$

$\widetilde{\mathcal{L}} := (a^\top \tilde{x}) \cdot (b^\top \tilde{x})$

$\widehat{\mathcal{L}} := (a^\top \hat{x}) \cdot (b^\top \hat{x})$

if $\widetilde{\mathcal{L}} \geq \widehat{\mathcal{L}}$ **then**

  $\mathcal{L} := \widetilde{\mathcal{L}}$, $x := \tilde{x}$

else

  $\mathcal{L} := \widehat{\mathcal{L}}$, $x := \hat{x}$

Output: lower bound $\mathcal{L}$ for (Ro.Cc) and corresponding solution x

In addition to the polynomial time complexity, Algorithm 5.1 yields a constant approximation ratio for (Ro.Cc).

Theorem 5.9 *Algorithm 5.1 is a polynomial time* 4.5*-approximation algorithm for the cardinality constrained rectangular knapsack problem.*

Proof. Algorithm 5.1 returns a feasible solution in polynomial time (see Lemma 5.8).

Case 1: k even

Since the coefficients a_i, $b_{\pi(j)}$ are in non-increasing order, it holds that

$$\begin{aligned} \mathcal{U} &= \sum_{i=1}^{k} \sum_{j=1}^{k} a_i b_{\pi(j)} \\ &= \sum_{i=1}^{\kappa} \sum_{j=1}^{\kappa} a_i b_{\pi(j)} + \sum_{i=\kappa+1}^{k} \sum_{j=1}^{\kappa} a_i b_{\pi(j)} + \sum_{i=1}^{\kappa} \sum_{j=\kappa+1}^{k} a_i b_{\pi(j)} + \sum_{i=\kappa+1}^{k} \sum_{j=\kappa+1}^{k} a_i b_{\pi(j)} \\ &\leq 4 \cdot \sum_{i=1}^{\kappa} \sum_{j=1}^{\kappa} a_i b_{\pi(j)} \leq 4\mathcal{L} \end{aligned}$$

Case 2: k odd

In analogy to case 1 we again use the fact that the coefficients a_i, $b_{\pi(j)}$ are in

non-increasing order. We can assume without loss of generality that:

$$\sum_{i=1}^{\underline{\kappa}}\sum_{j=1}^{\overline{\kappa}} a_i b_{\pi(j)} \leq \sum_{i=1}^{\overline{\kappa}}\sum_{j=1}^{\underline{\kappa}} a_i b_{\pi(j)}.$$

This inequality is equivalent to:

$$\begin{aligned}
& \sum_{i=1}^{\underline{\kappa}}\sum_{j=1}^{\underline{\kappa}} a_i b_{\pi(j)} + \sum_{i=1}^{\underline{\kappa}} a_i b_{\pi(\overline{\kappa})} \leq \sum_{i=1}^{\underline{\kappa}}\sum_{j=1}^{\underline{\kappa}} a_i b_{\pi(j)} + \sum_{j=1}^{\underline{\kappa}} a_{\overline{\kappa}} b_{\pi(j)} \\
\Leftrightarrow \quad & \sum_{i=1}^{\overline{\kappa}} a_i b_{\pi(\overline{\kappa})} - a_{\overline{\kappa}} b_{\pi(\overline{\kappa})} \leq \sum_{j=1}^{\overline{\kappa}} a_{\overline{\kappa}} b_{\pi(j)} - a_{\overline{\kappa}} b_{\pi(\overline{\kappa})} \\
\Leftrightarrow \quad & \sum_{i=1}^{\overline{\kappa}} a_i b_{\pi(\overline{\kappa})} \leq \sum_{j=1}^{\overline{\kappa}} a_{\overline{\kappa}} b_{\pi(j)}. \qquad (5.6)
\end{aligned}$$

Thus, the following inequality holds. Note that we use Equations 5.4 and 5.5 to classify several terms.

$$\begin{aligned}
\mathcal{U} = & \sum_{i=1}^{k}\sum_{j=1}^{k} a_i b_{\pi(j)} \\
= & \sum_{i=1}^{\overline{\kappa}}\sum_{j=1}^{\underline{\kappa}} a_i b_{\pi(j)} + \sum_{i=\overline{\kappa}+1}^{k}\sum_{j=\underline{\kappa}+1}^{k} a_i b_{\pi(j)} + \sum_{i=1}^{\overline{\kappa}}\sum_{j=\underline{\kappa}+1}^{k} a_i b_{\pi(j)} + \sum_{i=\overline{\kappa}+1}^{k}\sum_{j=1}^{\underline{\kappa}} a_i b_{\pi(j)} \\
\leq & \mathcal{L} + \sum_{i=1}^{\underline{\kappa}}\sum_{j=1}^{\overline{\kappa}} a_i b_{\pi(j)} + \sum_{i=1}^{\overline{\kappa}}\sum_{j=1}^{\overline{\kappa}} a_i b_{\pi(j)} + \sum_{i=1}^{\underline{\kappa}}\sum_{j=1}^{\underline{\kappa}} a_i b_{\pi(j)} \\
\leq & 2\mathcal{L} + \left(\sum_{i=1}^{\overline{\kappa}}\sum_{j=1}^{\underline{\kappa}} a_i b_{\pi(j)} + \sum_{i=1}^{\overline{\kappa}} a_i b_{\pi(\overline{\kappa})}\right) + \sum_{i=1}^{\underline{\kappa}}\sum_{j=1}^{\underline{\kappa}} a_i b_{\pi(j)} \\
\overset{(5.6)}{\leq} & 3\mathcal{L} + \sum_{j=1}^{\overline{\kappa}} a_{\overline{\kappa}} b_{\pi(j)} + \sum_{i=1}^{\underline{\kappa}}\sum_{j=1}^{\underline{\kappa}} a_i b_{\pi(j)} \\
\leq & 3\mathcal{L} + a_{\overline{\kappa}} b_{\pi(\overline{\kappa})} + \sum_{j=1}^{\underline{\kappa}} a_{\overline{\kappa}} b_{\pi(j)} + \sum_{i=1}^{\underline{\kappa}}\sum_{j=1}^{\underline{\kappa}} a_i b_{\pi(j)} \\
= & 3\mathcal{L} + a_{\overline{\kappa}} b_{\pi(\overline{\kappa})} + \sum_{i=1}^{\overline{\kappa}}\sum_{j=1}^{\underline{\kappa}} a_i b_{\pi(j)} \\
\leq & 4\mathcal{L} + a_{\overline{\kappa}} b_{\pi(\overline{\kappa})} \quad \text{// worst case: } a_{\overline{\kappa}} b_{\pi(\overline{\kappa})} = a_i b_{\pi(j)},\ i = 1, \ldots, \overline{\kappa},\ j = 1, \ldots, \underline{\kappa} \\
\leq & 4\mathcal{L} + \frac{1}{\overline{\kappa}} \cdot \frac{1}{\underline{\kappa}} \cdot \sum_{i=1}^{\overline{\kappa}}\sum_{j=1}^{\underline{\kappa}} a_i b_{\pi(j)} \\
\leq & 4\mathcal{L} + \frac{1}{\overline{\kappa}} \cdot \frac{1}{\underline{\kappa}} \cdot \mathcal{L}
\end{aligned}$$

$$\leq \left(4 + \frac{1}{\overline{\kappa}} \cdot \frac{1}{\underline{\kappa}}\right) \cdot \mathcal{L} \tag{5.7}$$
$$\leq 4.5\mathcal{L}$$

In summary, this yields the approximation factor:

$$\max\left(\frac{\mathcal{L}}{OPT}, \frac{OPT}{\mathcal{L}}\right) \leq \max\left(\frac{\mathcal{L}}{\mathcal{U}}, \frac{\mathcal{U}}{\mathcal{L}}\right) \leq \frac{4.5 \cdot \mathcal{L}}{\mathcal{L}} = 4.5.$$

□

As presented in the proof of Theorem 5.9, we can guarantee better results for even values of k. Also, for odd values of k, the quality of the approximation increases for increasing k.

Remark 5.10

- *If k is even, the result of Theorem 5.9 improves to a* 4*-approximation algorithm.*
- *For fixed odd values of k, Algorithm 5.1 is a polynomial time ρ-approximation algorithm for* (Ro.Cc) *with (cf. Equation* (5.7)*):*

k	3	5	7	9	11	13	15	17	19
$\underline{\kappa}$	1	2	3	4	5	6	7	8	9
$\overline{\kappa}$	2	3	4	5	6	7	8	9	10
$\rho = 4 + \frac{1}{\overline{\kappa}} \cdot \frac{1}{\underline{\kappa}}$	$\frac{9}{2}$	$\frac{25}{6}$	$\frac{49}{12}$	$\frac{81}{20}$	$\frac{121}{30}$	$\frac{169}{42}$	$\frac{225}{56}$	$\frac{289}{72}$	$\frac{361}{90}$

5.3.1 Improved approximation algorithm

In practice, we can formulate an improved variant of Algorithm 5.1. Due to the definition of the lower bound solution $\tilde{x}$ (see Equation 5.2), we do not use the full capacity of the cardinality constraint of (Ro.Cc) if the sets $\{1, \ldots, \overline{\kappa}\}$ and $\{\pi(1), \ldots, \pi(\underline{\kappa})\}$ are not disjoint, i. e., if $\sum_{i=1}^{n} \tilde{x}_i < k$. Hence, it is possible to increase the lower bound value by including further items. Algorithm 5.2 demonstrates a possible procedure to compute an improved lower bound $\mathcal{L}_{\text{impr}}$ that takes this in consideration.

An additional parameter k', which we name *adaptive capacity*, is introduced to increase the sets $\{1, \ldots, \overline{\kappa}\}$ and $\{\pi(1), \ldots, \pi(\underline{\kappa})\}$, and, therefore, increase the number of selected items, without violating the constraint. At first, k' is set to k. After computing the lower bound solution $\tilde{x}$ as defined in (5.2), the algorithm tests whether k items are selected or not. In the latter case, the adaptive capacity k' is increased by the difference $k - \sum_{i=1}^{n} \tilde{x}_i$. A re-computation of $\tilde{x}$, using k' as capacity, allows to include more items in accordance with the ordering of the respective coefficients a_i or $b_{\pi(i)}$ which compensates for the fact

that the original sets are not disjoint. Subsequently, it is tested again if the constraint is satisfied with equality. If not, the adaptive capacity k' is further increased. Otherwise, the algorithm continues by computing $\hat{x}$ using the current value of the parameter k' as capacity and testing which of the lower bound solutions is superior.

Algorithm 5.2 Improved approximation algorithm for (Ro.Cc)

Input: coefficients $a = (a_1, .., a_n)^\top$, $b = (b_1, \ldots, b_n)^\top$, capacity k

1: $\tilde{x} := \mathbf{0}_n$, $\hat{x} := \mathbf{0}_n$, stop$:= 0$, $k' := k$, $a := 1$

2: compute permutation $\pi \in \mathcal{S}_n$ such that

$$\begin{cases} b_{\pi(j)} > b_{\pi(j+1)}, \text{ or} \\ b_{\pi(j)} = b_{\pi(j+1)} \text{ and } a_{\pi(j)} \geq a_{\pi(j+1)} \end{cases} \quad \text{for } j = 1, \ldots, n-1$$

3: **while** stop $= 0$ **do**

4: **for** $i := a, \ldots, \left\lfloor \frac{k'}{2} \right\rfloor$ **do** // *include further items*

5: $\tilde{x}_i := 1$, $\tilde{x}_{\pi(i)} := 1$

6: $\tilde{x}_{\left\lceil \frac{k'}{2} \right\rceil} := 1$

7: **if** $\sum_{i=1}^{n} \tilde{x}_i < k$ **then** // *no equality in constraint*

8: $a := \left\lfloor \frac{k'}{2} \right\rfloor + 1$

9: $k' := k' + \left(k - \sum_{i=1}^{n} \tilde{x}_i\right)$ // *increase adaptive capacity k'*

10: **else** // *equality obtained*

11: stop $:= 1$

12: **for** $i := 1, \ldots, \left\lfloor \frac{k'}{2} \right\rfloor$ **do** // *compute $\hat{x}$*

13: $\hat{x}_i := 1$, $\hat{x}_{\pi(i)} := 1$

14: $\hat{x}_{\pi\left(\left\lceil \frac{k'}{2} \right\rceil\right)} := 1$

15: $\widetilde{\mathcal{L}} := (a^\top \tilde{x}) \cdot (b^\top \tilde{x})$

16: $\widehat{\mathcal{L}} := (a^\top \hat{x}) \cdot (b^\top \hat{x})$

17: **if** $\widetilde{\mathcal{L}} \geq \widehat{\mathcal{L}}$ **then**

18: $\mathcal{L} := \widetilde{\mathcal{L}}$, $x := \tilde{x}$

19: **else**

20: $\mathcal{L} := \widehat{\mathcal{L}}$, $x := \hat{x}$

Output: lower bound $\mathcal{L}$ and corresponding solution x

Lemma 5.11 *If, in Step 9 of Algorithm 5.2, the solution $\tilde{x}$ allows to increase the adaptive capacity k' to $k' + (k - \sum_{i=1}^{n} \tilde{x}_i)$, this is also feasible for the computation of $\hat{x}$.*

Proof. For ease of notation we assume that we are examining the iteration where the adaptive capacity k' is increased for the first time from the capacity k to $k' = k + (k - \sum_{i=1}^{n} \tilde{x}_i)$. The following discussion can be applied in an analogous manner to all further iterations by adapting the notation accordingly.

If k is even, we know that $\tilde{x} = \hat{x}$ and the statement is trivially true. Otherwise, i. e., if k is odd, we can take advantage of the fact that the solution $\tilde{x}$ or $\hat{x}$ uses less than k items if:

- for $\tilde{x}$: $\{1, \dots, \overline{\kappa}\} \cap \{\pi(1), \dots, \pi(\underline{\kappa})\} \neq \emptyset$.
- for $\hat{x}$: $\{1, \dots, \underline{\kappa}\} \cap \{\pi(1), \dots, \pi(\overline{\kappa})\} \neq \emptyset$.

Therefore, we define

$$\begin{aligned}\tilde{\mathcal{I}} &:= \{1, \dots, \overline{\kappa}\} \cup \{\pi(1), \dots, \pi(\underline{\kappa})\},\\ \hat{\mathcal{I}} &:= \{1, \dots, \underline{\kappa}\} \cup \{\pi(1), \dots, \pi(\overline{\kappa})\} \text{ and}\\ \mathcal{J} &:= \tilde{\mathcal{I}} \cap \hat{\mathcal{I}} = \{1, \dots, \underline{\kappa}\} \cup \{\pi(1), \dots, \pi(\underline{\kappa})\}.\end{aligned}$$

It holds that $\sum_{i=1}^{n} \tilde{x}_i = |\tilde{\mathcal{I}}|$ and that $\sum_{i=1}^{n} \hat{x}_i = |\hat{\mathcal{I}}|$. Furthermore, we know that

$$|\tilde{\mathcal{I}}| = \begin{cases} |\mathcal{J}| & \text{, if } \overline{\kappa} \in \{\pi(1), \dots, \pi(\underline{\kappa})\}, \text{i, e., if } \tilde{\mathcal{I}} = \mathcal{J} \\ |\mathcal{J}| + 1 & \text{, else} \end{cases}$$

Furthermore, we know that

$$|\hat{\mathcal{I}}| = \begin{cases} |\mathcal{J}| & \text{, if } \pi(\overline{\kappa}) \in \{1, \dots, \underline{\kappa}\}, \text{i, e., if } \hat{\mathcal{I}} = \mathcal{J} \\ |\mathcal{J}| + 1 & \text{, else} \end{cases}$$

Considering these relations, we distinguish four cases:

Case 1: $|\tilde{\mathcal{I}}| = |\mathcal{J}| \wedge |\hat{\mathcal{I}}| = |\mathcal{J}|$
Thus, k' can be set to $k + (k - |\mathcal{J}|) = k + (k - |\tilde{\mathcal{I}}|)$ for $\tilde{x}$ and for $\hat{x}$.

Case 2: $|\tilde{\mathcal{I}}| = |\mathcal{J}| + 1 \wedge |\hat{\mathcal{I}}| = |\mathcal{J}| + 1$
Thus, k' can be set to $k + (k - (|\mathcal{J}| + 1)) = k + (k - |\tilde{\mathcal{I}}|)$ for $\tilde{x}$ and for $\hat{x}$.

Case 3: $|\tilde{\mathcal{I}}| = |\mathcal{J}| + 1 \wedge |\hat{\mathcal{I}}| = |\mathcal{J}|$
For $\hat{x}$, k' can be set to $k + (k - |\mathcal{J}|) = k + (k - |\hat{\mathcal{I}}|)$. Defined by $\tilde{x}$, k' will be set to $k + (k - |\tilde{\mathcal{I}}|) = k + (k - (|\mathcal{J}| + 1)) < k + (k - |\mathcal{J}|)$ which is feasible for $\hat{x}$.

Case 4: $|\tilde{\mathcal{I}}| = |\mathcal{J}| \wedge |\hat{\mathcal{I}}| = |\mathcal{J}| + 1$
Since $|\tilde{\mathcal{I}}| = |\mathcal{J}|$, we know that $\overline{\kappa} \in \{\pi(1), \dots, \pi(\underline{\kappa})\}$ $(*)$. In a first iteration we examine the consequences of setting $k' := k + 1$. Thus, k' is even and we define the corresponding solution as:

$$x'_i = \begin{cases} 1 & \text{, for } i \in \{1, \dots, \overline{\kappa}\} \cup \{\pi(1), \dots, \pi(\overline{\kappa})\} \\ 0 & \text{, otherwise} \end{cases},$$

where $\{1, \dots, \overline{\kappa}\} \cup \{\pi(1), \dots, \pi(\overline{\kappa})\} \overset{(*)}{=} \{1, \dots, \underline{\kappa}\} \cup \{\pi(1), \dots, \pi(\overline{\kappa})\} = \hat{\mathcal{I}}$. Thus, setting the adaptive capacity k' to $k + 1$ does not change $\hat{x}$, i. e., $x' = \hat{x}$.

Hence, k' can be set to

$$k + 1 + (k - (|\hat{\mathcal{I}}| + 1)) = k + (k - |\mathcal{J}|) = k + (k - |\tilde{\mathcal{I}}|)$$

for $\tilde{x}$ and for $\hat{x}$.

□

Lemma 5.12 *Let n be the number of items and let k be the capacity of* (Ro.Cc)*. Algorithm 5.2 terminates and has a worst case time complexity of $\mathcal{O}(n \log n)$.*

Proof. Critical for the termination of Algorithm 5.2 is the while-loop for computing the solution $\tilde{x}$ with $\sum_{i=1}^{n} \tilde{x}_i = k$. In the first iteration, at least $\overline{\kappa}$ variables are set to 1. The parameter k' is increased by at least 1 in each consecutive iteration and, thus, in at least every second iteration an additional entry of $\tilde{x}$ is set to 1. Hence, after at most $2 \cdot \underline{\kappa} + 1 = k$ iterations k variables have been selected for $\tilde{x}$ and the loop terminates.

We take advantage of the ordering of the coefficients to set only new variables to 1 if the adaptive capacity is increased. Thus, the execution of the while loop requires $\mathcal{O}(k)$. The complexity of Algorithm 5.2 is, as the complexity of Algorithm 5.1, determined by the sorting algorithm, i. e., Algorithm 5.2 has a worst case time complexity of $\mathcal{O}(n \log n)$. □

Example 5.13 *We apply the improved approximation algorithm, Algorithm 5.2, on the instance of* (Ro.Cc) *of Example 5.7. The solution $\tilde{x}$ is defined by the set*

$$\tilde{\mathcal{I}} = \{1, 2, 3\} \cup \{\pi(1), \pi(2)\} = \{1, 2, 3\} \cup \{2, 4\} = \{1, 2, 3, 4\}$$

with $|\tilde{\mathcal{I}}| = \sum_{i=1}^{n} \tilde{x}_i = 4 < 5$. Thus, the adaptive capacity can be set to $k' := 5 + (5 - 4) = 6$. The re-computation of $\tilde{x}$ leads to

$$\tilde{\mathcal{I}} = \{1, 2, 3\} \cup \{\pi(1), \pi(2), \pi(3)\} = \{1, 2, 3\} \cup \{2, 4, 6\} = \{1, 2, 3, 4, 6\}$$

with $|\tilde{\mathcal{I}}| = \sum_{i=1}^{n} \tilde{x}_i = 5$. Hence, the cardinality constraint is tight and, since k' is even, the solution $x = (1, 1, 1, 1, 0, 1, 0, 0)^\top$ generating the improved lower bound $\mathcal{L} = f(x) = 920$ and approximating the optimal solution is found. The optimal solution of the instance is $x^* = (1, 1, 1, 1, 0, 1, 0, 0)^\top$ and, thus, identical to the improved lower bound solution.

As proven above, setting the adaptive capacity to $k' := 6$ is also feasible for $\hat{x}$. The solution $\hat{x}$ defined by $k' = 5$ corresponds to the set

$$\hat{\mathcal{I}} = \{1, 2\} \cup \{\pi(1), \pi(2), \pi(3)\} = \{1, 2\} \cup \{2, 4, 6\} = \{1, 2, 4, 6\}$$

with $|\hat{\mathcal{I}}| = \sum_{i=1}^{n} \hat{x}_i = 4 < 5$. Hence, one additional item can be included resulting in the same lower bound solution $x = (1, 1, 1, 1, 0, 1, 0, 0)^\top$ again (with $k' = 6$). The areas of rectangles corresponding to the lower bounds $\widetilde{\mathcal{L}}$ and $\widehat{\mathcal{L}}$ based on the first computations of $\tilde{x}$ and $\hat{x}$ (Algorithm 5.1), respectively, and the improved lower bound $\mathcal{L}$ (Algorithm 5.2) are shown in Figure 5.4.

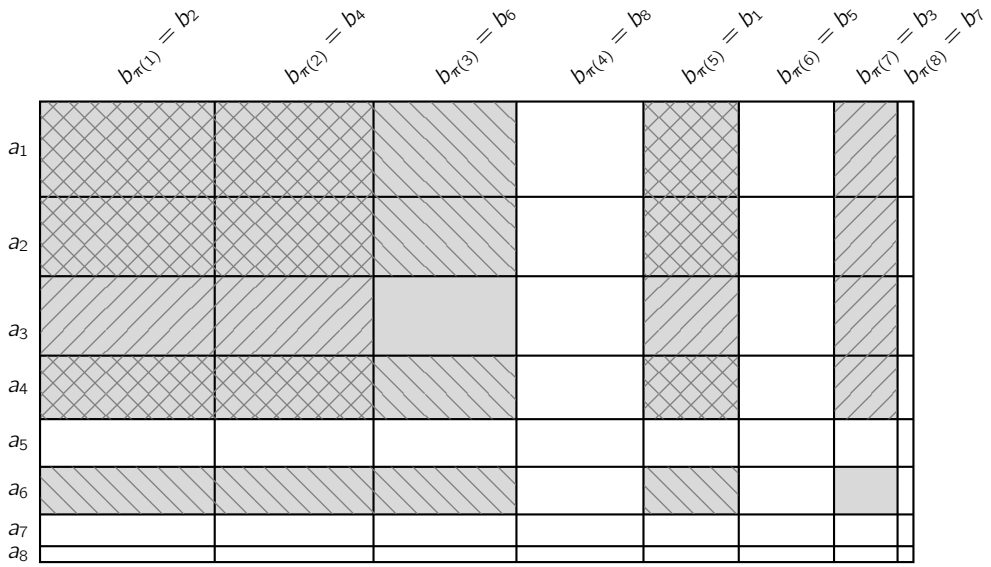


Figure 5.4: Lower bounds $\widetilde{\mathcal{L}}$ (▨) , $\widehat{\mathcal{L}}$ (▧) and $\mathcal{L}$ (▩) for Examples 5.7 and 5.13.

5.4 Computational results

Computational experiments were performed on an Intel Quadcore 2,80 GHz with 4 GB RAM. We implemented both approximation Algorithms 5.1 and 5.2 in C. The instances were randomly generated using the code of Pisinger [2016], slightly adapting it to the special structure of the profits p_{ij} and to the capacity constraint. The number of items n was chosen up to 400 and the coefficients a_i, b_i were restricted to the range $[1, n/2]$. For each size of the problem, three different constraint slacknesses c_k (recall: $c_k \cdot \sum_{i=1}^{n} 1 = k$) were chosen: $c_k = 0.25$, $c_k = 0.5$, and $c_k = 0.75$. For every class of instances, the presented results are the average over 10 instances.

Table 5.1 presents the averaged approximation ratios $z^*/\mathcal{L}$ and $z^*/\mathcal{L}_{\text{impr}}$. The optimal objective function value z^* was computed by using the exact solution algorithm of Caprara et al. [1999] [code downloaded from the web page of Pisinger, 2016]. The results indicate for general instances, that in practice the approximation ratio of both algorithms is much better than the guaranteed ratio of 4.5 and that the improved algorithm, Algorithm 5.2, yields even better results than the basic version, Algorithm 5.1. It is interesting to notice, on the one hand, that the approximation quality of Algorithm 5.1 becomes worse the larger the constraint slackness is. This is most likely because, having a large constraint slackness, it becomes more likely that items are chosen due to both orderings. On the other hand, the approximation of Algorithm 5.2 becomes better because this weakness is compensated in this algorithm and a larger capacity value k improves the approximation (see Remark 5.10).

In practice, the quality of the approximation can only be evaluated by the ratio $\mathcal{U}/\mathcal{L}$, and $\mathcal{U}/\mathcal{L}_{\text{impr}}$, respectively, where an upper bound $\mathcal{U}$ can be computed by the scheme of

		Algorithm 5.1		Algorithm 5.2	
n	k	$z^*/\mathcal{L}$	$\mathcal{U}/\mathcal{L}$	$z^*/\mathcal{L}_{\text{impr}}$	$\mathcal{U}/\mathcal{L}_{\text{impr}}$
100	25	1.36	1.77	1.20	1.56
	50	1.31	1.65	1.06	1.34
	75	1.38	1.56	1.02	1.15
200	50	1.35	1.77	1.17	1.52
	100	1.33	1.66	1.07	1.34
	150	1.37	1.54	1.02	1.14
300	75	1.34	1.76	1.18	1.55
	150	1.33	1.69	1.06	1.35
	225	1.35	1.52	1.01	1.14
400	100	1.33	1.75	1.18	1.54
	200	1.32	1.66	1.06	1.33
	300	1.37	1.56	1.02	1.15

Table 5.1: Approximation ratios $\rho = z^*/\mathcal{L}$ and $\rho = z^*/\mathcal{L}_{\text{impr}}$ and ratios $\mathcal{U}/\mathcal{L}$ and $\mathcal{U}/\mathcal{L}_{\text{impr}}$ of Algorithms 5.1 and 5.2 for instances of (Ro.Cc) of different sizes.

Lemma 5.4. The results do also indicate, that, for general instances, the approximation is far better than the guaranteed ratio suggests. Furthermore, the approximations are computed very fast: Both approximation algorithms require 0.01 seconds or less per instance.

5.5 Hypervolume maximizing representations

In the introduction of this chapter we presented the cardinality constrained bi-objective knapsack problem (2o.Cc). The objective function of (Ro.Cc) is determined by the product of the two objective functions of (2o.Cc). The constraint is equal in both problems. Therefore, a feasible solution x of (2o.Cc) is also feasible for (Ro.Cc). The objective function value of (Ro.Cc) yields the hypervolume corresponding to the solution x in (2o.Cc).

An optimal solution x^* of (Ro.Cc) is an efficient solution of (2o.Cc) that maximizes the hypervolume. It is, therefore, a particular representative solution of (2o.Cc) where both objective function values are treated as equally important. Thus, Algorithms 5.1 and 5.2 approximate the optimal hypervolume, considering $(0,0)^\top$ as reference point, and also identify a feasible solution $\tilde{x}$ of (2o.Cc) and the corresponding point $\left(a^\top \tilde{x}, b^\top \tilde{x}\right)$ that realizes this approximation.

In Figure 5.5 the nondominated set of an instance of (2o.Cc) is shown. The nondominated point maximizing the hypervolume as well as the point obtained by using the approximation (Algorithm 5.1 or 5.2) are plotted for this example. Note that, in contrast to this example, it cannot be assumed in general that the approximating point is nondominated. Furthermore, the example does nicely illustrate that the upper bound $\mathcal{U}$ corresponds

to the hypervolume of the ideal point of (2o.Cc). The upper bound is computed by separately choosing the k best items for each objective function, i. e., by building individual optima.

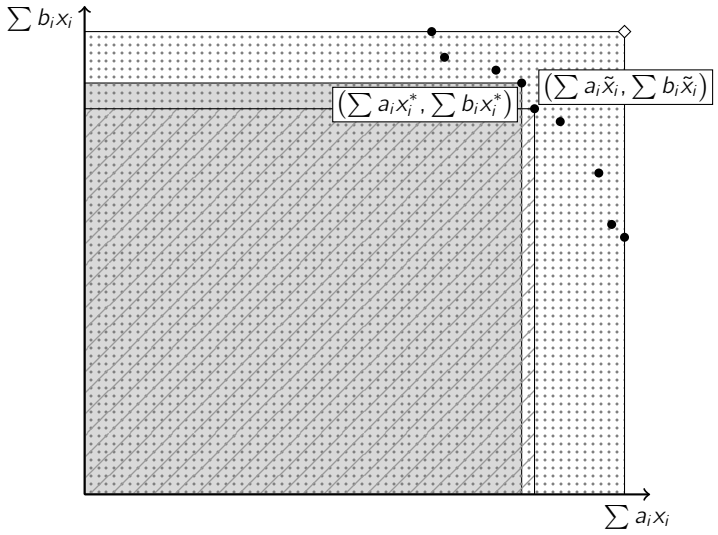


Figure 5.5: Nondominated points of an instance of (2o.Cc), the ideal point $\diamond$, and the upper bound $\mathcal{U}$(▦) and lower bound $\mathcal{L}$(▨) on the maximal hypervolume (■) obtained when choosing one feasible point.

5.5.1 Shifting the reference point

The model of (Ro.Cc) is associated to $(0, 0)^\top$ as reference point. The nadir point may also be an appropriate choice as reference point and an approach for computing an approximation of a representative set that we present in Section 5.5.2 will require several choices of reference points. Thus, we analyze a shift of the reference point $(R_1, R_2)^\top$. This necessitates a more precise formulation of the aim of optimization: A solution is sought that is feasible for (2o.Cc), that dominates the reference point and that maximizes the hypervolume defined by the corresponding point and the reference point. The second condition has to be enforced, since otherwise points that are dominated by the reference point may also generate a positive hypervolume w. r. t. $(R_1, R_2)^\top$, cf. Figure 5.6 for an example.

We generalize the formulation of (Ro.Cc) to an arbitrary reference point $(R_1, R_2)^\top \in \mathbb{R}^2$. At first, the constants R_1 and R_2 are subtracted in the first and second objective function of (2o.Cc), respectively. The product of both objective functions

$$\left(\sum_{i=1}^{n} a_i x_i - R_1\right) \cdot \left(\sum_{i=1}^{n} b_i x_i - R_2\right)$$

determines the hypervolume spanned by the chosen point $(\sum_{i=1}^{n} a_i x_i, \sum_{i=1}^{n} b_i x_i)$ and the

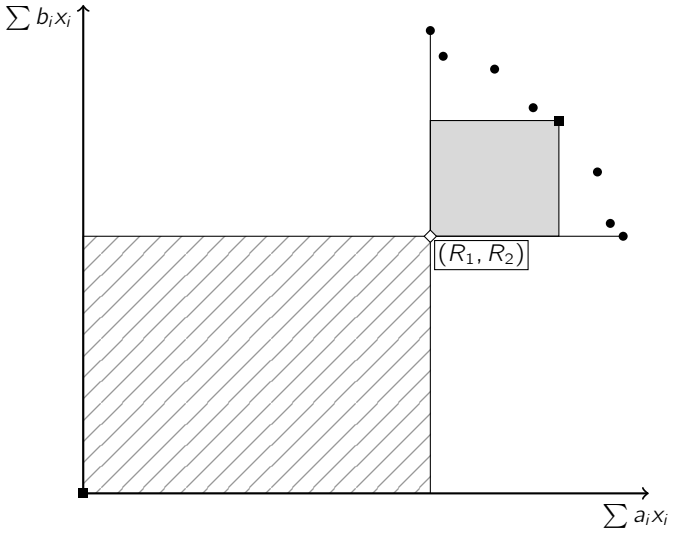


Figure 5.6: Nondominated points of an instance of (2o.Cc), the nadir point $\diamond$ as reference point (R_1, R_2) and two feasible points (■) of (2o.Cc) generating a positive hypervolume (▨ and ■).

reference point (R_1, R_2), cf. Figure 5.6. Furthermore, two additional constraints have to be introduced to ensure that the reference point is dominated by a feasible point of the generalized model:

$$\sum_{i=1}^{n} a_i x_i \geq R_1 \quad \text{and} \quad \sum_{i=1}^{n} b_i x_i \geq R_2.$$

To obtain a tractable problem formulation, the shift of the overall problem has to be translated to the single items. We suggest two alternative approaches:

1. The product of the two objective functions of (2o.Cc) includes a quadratic term, which equals the objective function of (Ro.Cc), and a linear term:

$$\left(\sum_{i=1}^{n} a_i x_i - R_1\right) \cdot \left(\sum_{i=1}^{n} b_i x_i - R_2\right)$$
$$= \sum_{i=1}^{n}\sum_{j=1}^{n} a_i b_j \, x_i x_j - \sum_{i=1}^{n} (a_i \cdot R_2 + b_i \cdot R_1) x_i + R_1 R_2.$$

   This can be interpreted in the following way: A shift of the reference point does change the profit of including a certain item. The original profit $a_i b_i$ is reduced by $a_i \cdot R_2 + b_i \cdot R_1$, which may result in a negative profit for including item i. In contrast, the profits for choosing pairs of items remain unchanged.

2. We may take advantage of the following fact: It is easy to see that equality can be assumed for the cardinality constraint of (2o.Cc) without changing the set of nondominated points. Since all profits are positive, the selection of additional items can

never decrease the value of an objective function. By enforcing equality, subtracting a constant R in one objective function can be passed down to the coefficients by subtracting $\frac{R}{k}$ from each coefficient. By selecting exactly k items, a value of $k \cdot \frac{R}{k} = R$ is subtracted from the sum of the original coefficients. We define $\bar{a}_i := a_i - \frac{R_1}{k}$ and $\bar{b}_i := b_i - \frac{R_2}{k}$, for $i = 1, \ldots, n$. Hence, for every feasible solution including k items it holds:

$$\sum_{i=1}^{n} \bar{a}_i x_i = \sum_{i=1}^{n} a_i x_i - R_1 \quad \text{and} \quad \sum_{i=1}^{n} \bar{b}_i x_i = \sum_{i=1}^{n} b_i x_i - R_2$$

Thus, a possible model for computing the maximal hypervolume with a shifted reference point can be the following:

$$\begin{aligned}
\max \quad & \left(\sum_{i=1}^{n} \bar{a}_i x_i\right) \cdot \left(\sum_{i=1}^{n} \bar{b}_i x_i\right) = \sum_{i=1}^{n}\sum_{j=1}^{n} \bar{a}_i \bar{b}_j \, x_i x_j \\
\text{s.t.} \quad & \sum_{i=1}^{n} x_i = k \\
& \sum_{i=1}^{n} \bar{a}_i x_i \geq 0 \\
& \sum_{i=1}^{n} \bar{b}_i x_i \geq 0 \\
& x_i \in \{0, 1\}, \qquad i = 1, \ldots, n.
\end{aligned} \tag{QP}$$

Note that R_1 and R_2 have been subtracted on both sides of the second and third constraint, respectively.

(QP) has a similar structure as (Ro.Cc). One difference is, that the coefficients $\bar{a}_i$ and $\bar{b}_i$ are generally in $\mathbb{Z}$. By allowing negative coefficients, the lower bound solutions introduced above may become infeasible: Algorithms 5.1 and 5.2 do not test whether the coefficients $\bar{a}_{\pi(i)}$ and $\bar{b}_i$, for $i = 1, \ldots, \overline{\kappa}$, are negative. Therefore, it may happen that the computed lower bound solution $\tilde{x}$ generates values $\sum_{i=1}^{n} \bar{a}_i \tilde{x}_i < 0$ or $\sum_{i=1}^{n} \bar{b}_i \tilde{x}_i < 0$, i. e., that $\tilde{x}$ has a negative objective function value or that it corresponds to a feasible point of (2o.Cc) that is dominated by the reference point. In both cases $\tilde{x}$ is not feasible for (QP). As a consequence, new or further ideas are necessary to design an approximation algorithm for the remodeled problem.

If (QP) also includes a classical capacity constraint, for example instead of the cardinality constraint, it can be proven that no polynomial time approximation algorithm with fixed approximation ratio exists. The proof of Rader and Woeginger [2002] for the modified quadratic knapsack problem with positive and negative coefficients, which uses a reduction from Subset Sum ([SP3] in Garey and Johnson [1979]), already shows this. But for (QP), to the best of our knowledge, it is unclear whether such an approximation exists or not.

5.5.2 Representative set

Another interesting extension is to generalize (Ro.Cc) such that not only one representative point but a representative set of the efficient set of (2o.Cc) could be computed. We are searching for ℓ efficient solutions $x^1, \ldots, x^\ell \in \{0,1\}^n$, which maximize the hypervolume spanned by the points $(a^\top x^j, b^\top x^j)_{j=1,\ldots,\ell}$ and the origin. If we assume that the points $(a^\top x^j, b^\top x^j)_{j=1,\ldots,\ell}$ are ordered with respect to their first component $a^\top x^1 \geq \ldots \geq a^\top x^\ell$, the hypervolume dominated by these ℓ points can be computed as the hypervolume of the first point plus the incremental hypervolume given by the following points, or as the sum of individual hypervolumes of all points minus the intersection area of consecutive pairs of points:

$$(x^1)^\top a\, b^\top x^1 + \sum_{j=2}^{\ell} (x^j)^\top a\, b^\top (x^j - x^{j-1}) \tag{5.8}$$

$$= \sum_{j=1}^{\ell} (x^j)^\top a\, b^\top x^j - \sum_{j=2}^{\ell} (x^j)^\top a\, b^\top x^{j-1}. \tag{5.9}$$

In Figure 5.7, the hypervolume spanned by three nondominated points is presented. This selection is optimal, i. e., the points generate the maximal hypervolume spanned by three points and the origin, in this example.

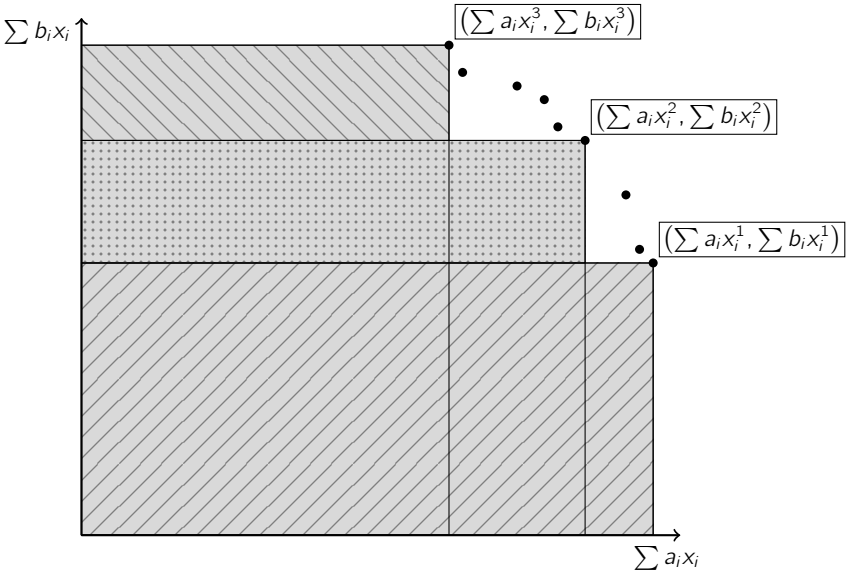


Figure 5.7: Hypervolume (▩) generated by three solutions x^1, x^2 and x^3. It can be composed by the three highlighted areas (▨, ▦, and ▧) using Equation (5.8).

The problem can be modeled as a quadratic integer programming problem (QIP) with respect to the variable vector $x^\top = ((x^1)^\top, \ldots, (x^\ell)^\top)$, $x \in \{0,1\}^{\ell n}$. We use Equation (5.9)

to formulate the quadratic objective function.

$$\begin{aligned} \max \quad & x^\top Q x \\ \text{s.t.} \quad & \mathbf{1}_n x^j \leq k, \qquad j = 1, \dots, \ell \\ & a^\top x^j \geq a^\top x^{j+1}, \qquad j = 1, \dots, \ell - 1 \\ & x \in \{0,1\}^{\ell n}, \end{aligned} \tag{QIP}$$

with objective matrix:

$$Q = \left(\begin{array}{c|c|c|c} a\,b^\top & & & \\ \hline -a\,b^\top & a\,b^\top & & \\ \hline & -a\,b^\top & a\,b^\top & \\ \hline & & \ddots & \ddots \end{array}\right).$$

The additional constraints ensure the ordering of points, which is necessary for the correct computation of the hypervolume, cf. Section 5.5.1. Omitting these constraints, it would be, for example, a good strategy to choose only two different solutions: for odd indices j set $x^j = x^*$, where x^* is the optimal solution for (Ro.Cc) with $\ell = 1$, and for even indices j set $x^j = \mathbf{0}_n$.

(QIP) has a very interesting structure and, furthermore, gives a closed formula for computing a representation of a bi-objective optimization problem w. r. t. the hypervolume. Therefore, it is very interesting for further research. Thinking about an approximation of (QIP), two alternative approaches may be promising. The first idea is to apply a greedy strategy by computing one representative after the other [see Guerreiro et al., 2016]. The first solution can be computed using (Ro.Cc), but, during the further procedure, the reference point has to be shifted to take the already computed information into account. So, it would be necessary to find an (efficient) approximation algorithm for (QP). The second idea is to go the other direction: First, compute an approximation for the bi-objective problem (2o.Cc) [see, for example, Bazgan et al., 2015], and, secondly, choose a representation that maximizes the hypervolume among this set. It is an interesting question, whether any quality guarantees can be given for this approach. We can conclude that there are a many open questions in this field, and that this is a promising and challenging line of research.

5.6 Conclusion and further ideas

In this chapter, we studied the cardinality constrained rectangular knapsack problem which is a variant of the quadratic knapsack problem. We presented a geometric interpretation

of this problem which motivates the denotation *rectangular* knapsack problem. Upper and lower bounds for the problem can be computed by sorting the coefficients of the objective function.

Motivated by the bound computations, we introduced an approximation algorithm for (Ro.Cc) that computes an approximate solution in polynomial time. Furthermore, we proved an approximation ratio of $\rho = 4.5$ for the algorithm. In practice, the algorithm can be further improved by selecting further items if the cardinality constraint is not met with equality. Thus, we also formulated an improved approximation algorithm.

We tested both algorithms on knapsack instances with up to 400 items and three different constraint slacknesses. The approximations were computed in 0.01 seconds or less per instance. We observed that in practice the approximation ratios of both algorithms are much better than the theoretically guaranteed ratio of 4.5. Thus, our approximation algorithms are an efficient tool to compute approximations of good quality for (Ro.Cc).

We also suggested a field of application for (Ro.Cc). Finding a representative solution of the bi-objective cardinality constrained knapsack problem that maximizes the hypervolume with the origin as reference point is modeled by the cardinality constrained rectangular knapsack problem. We presented further ideas for shifting the reference point and finding a representative set of solutions rather than only one solution. These are open questions and we think it is a promising and challenging line of research.

In the future it would be interesting to integrate the bound computations in a branch-and-bound procedure to formulate an exact algorithm for (Ro.Cc). Furthermore, the results of this chapter seem to be transferable to higher dimensions, where we think of problems of the form

$$\begin{aligned} \max \quad & f(x) = \prod_{j=1}^{m} \sum_{i=1}^{n} p_i^j x_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} x_i \leq k \\ & x_i \in \{0,1\}, \qquad i = 1, \ldots, n. \end{aligned}$$

The bound computations and algorithm formulations should be convertible without problems, whereas the proof of an approximation ratio may become more complicated due to more possible cases that may occur. Again, this problem models the search for a representative solution of the multi-objective cardinality constrained knapsack problem (mo.Cc) that maximizes the hypervolume with the origin as reference point and is, therefore, very interesting for future research.

6 Conclusion

In this thesis we investigated several variants of the classical knapsack problem. We showed that new perspectives on (multi-objective) knapsack problems, like more general formulations or reformulations, give valuable insights in structural properties, solution alternatives and algorithmic improvements.

We relaxed the multi-objective knapsack problem to the multi-objective unconstrained combinatorial optimization problem (MUCO). As we revealed, this problem has structural properties that are well-known in combinatorial geometry. The weight space decomposition of (MUCO) is defined by an arrangement of hyperplanes. In this way, we could specify a bound on the number of extreme supported solutions for (MUCO) depending on the number of objective functions and the number of items. We were able to prove that, for a fixed number of objectives, the number of extreme supported solutions is polynomially bounded with respect to the number of items. Furthermore, based on the structure of the arrangement of hyperplanes we described an efficient solution algorithm to compute the set of extreme supported solutions. We implemented a corresponding algorithm for a special variant of tri-objective problems. Our numerical test confirmed the theoretical results and showed that the algorithm runs very fast in practice.

A second variant of the multi-objective knapsack problem was obtained by considering integer, i. e., positive and negative objective function coefficients. This extension allows to model conflicting objectives even better than by using only positive coefficients. We introduced well-known algorithms that compute the set of extreme supported points for the knapsack problem. The findings about the (MUCO) problem can be used as a preprocessing to improve these algorithms. We implemented an algorithm for a special variant of tri-objective problems and described the algorithmic aspects. The computational tests showed that our preprocessing considerably speeds up the process.

For the bi-dimensional knapsack problem with one soft constraint we conversely used the bi-objective knapsack problem to generate solution alternatives. The bi-dimensional perspective only provides one optimal solution, whereas the bi-objective perspective allows to find this solution and further efficient solutions that are in a certain sense close to it. We transformed the soft constraint into an objective function. In doing so we again allow integer coefficients in the objective functions. The original objective function only contains positive coefficients whereas the newly generated objective function only contains negative coefficients. We applied a dynamic programming algorithm to solve the problem. The concept of dynamic programming is in general well-suited to solve knapsack problems that include negative objective function coefficients. We adapted two dominance relations

that are applied to prune states during the process. Further, we introduced a new pruning strategy that is based on the specific structure of the problem and on the search zones which are defined by supported solutions. The purpose of the transformation is to obtain additional information. However, as our computational tests indicate, a computation of the complete set of nondominated solutions generates too many solutions. We adapted the algorithm such that it focuses on nondominated points in a predefined region of interest in the objective space. Our numerical tests show that our bi-objective approach is considerably faster than an iterative application of a bi-dimensional approach as soon as several nondominated points are element of the region of interest. Our algorithm works very fast as long as the considered region of interest is small. For larger regions of interest the algorithm works in reasonable time as compared to the gained amount of information. We showed that the general idea of this approach (a transformation of soft constraints into objective functions) can also be applied for knapsack problems with more objectives or constraints and that a dynamic programming algorithm is suited to solve this problem. Our results on the computation of extreme supported points for multi-objective knapsack problems provides a useful tool for problems with more objective functions or more soft constraints.

The cardinality constrained rectangular knapsack problem is a combinatorial optimization problem that can be used to find a representative point for the bi-objective knapsack problem. This point maximizes the hypervolume indicator, which is a quality measure for the representation. We presented two approximation algorithms for the rectangular problem, where one is an improvement of the other. Both algorithms are polynomial time 4.5-approximation algorithms. We demonstrated by numerical tests that both approaches perform very well in practice. We also presented ideas that can be used to find a representative set of points for the bi-objective problem.

Concluding, we can state:

- The multi-objective unconstrained combinatorial optimization problem, a relaxed variant of the multi-objective knapsack problem, provides new interrelations to concepts of combinatorial geometry. This knowledge improves existing algorithms for multi-objective knapsack problems.
- The transformation of multi-dimensional knapsack problems including soft or uncertain constraints to multi-objective knapsack problems provides valuable alternatives to a decision-maker. This approach further emphasizes the great potential of multi-objective optimization for decision making processes.
- The rectangular knapsack problem, as a reformulated bi-objective knapsack problem, can be used to give a closed model to compute a specific representative solution. This is a new concept and a promising and challenging line for future research.

Bibliography

H. Aissi, A. R. Mahjoub, S. T. McCormick, and M. Queyranne. Strongly polynomial bounds for multiobjective and parametric global minimum cuts in graphs and hypergraphs. *Mathematical Programming*, 154(1):3–28, 2015. (Page 32)

Y. P. Aneja and K. P. K. Nair. Bicriteria transportation problem. *Management Science*, 25(1):73–78, 1979. (Pages 16, 74, 95, 97)

E. Balas and E. Zemel. An algorithm for large zero-one knapsack problems. *Operations Research*, 28(5):1130–1154, 1980. (Page 22)

C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, 22(4):469–483, 1996. (Pages 73, 74)

C. Bazgan, H. Hugot, and D. Vanderpooten. Implementing an efficient fptas for the 0–1 multi-objective knapsack problem. *European Journal of Operational Research*, 198(1): 47–56, 2009a. (Page 26)

C. Bazgan, H. Hugot, and D. Vanderpooten. Solving efficiently the 0–1 multi-objective knapsack problem. *Computers & Operations Research*, 36(1):260–279, 2009b. (Pages 24, 26, 98)

C. Bazgan, F. Jamain, and D. Vanderpooten. Approximate Pareto sets of minimal size for multi-objective optimization problems. *Operations Research Letters*, 43(1):1–6, 2015. (Page 138)

R. Beier and B. Vöcking. Random knapsack in expected polynomial time. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 232–241, San Diego, 2003. ACM. (Page 21)

R. Beier and B. Vöcking. An experimental study of random knapsack problems. *Algorithmica*, 45:121–136, 2006. (Page 92)

R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, 1957. (Page 21)

H. P. Benson and E. Sun. Outcome space partition of the weight set in multiobjective linear programming. *Journal of Optimization Theory and Applications*, 105(1):17–36, 2000. (Page 35)

H. P. Benson and E. Sun. A weight set decomposition algorithm for finding all efficient extreme points in the outcome set of a multiple objective linear program. *European Journal of Operational Research*, 139:26–41, 2002. (Page 72)

J. L. Bentley and T. A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Transactions on Computers*, C-28(9):643–647, 1979. (Pages 59, 62, 63, 66, 69, 83)

A. Billionnet and E. Soutif. An exact method based on Lagrangian decomposition for the 0–1 quadratic knapsack problem. *European Journal of Operational Research*, 157: 565–575, 2004. (Page 118)

A. Billionnet, A. Faye, and E. Soutif. A new upper bound for the 0–1 quadratic knapsack problem. *European Journal of Operational Research*, 112:664–672, 1999. (Page 118)

C. Blair. Sensitivity analysis for knapsack problems: a negative result. *Discrete Applied Mathematics*, 81:133–139, 1998. (Page 91)

H. L. Bodlaender and A. M. C. A. Koster. Combinatorial optimization on graphs of bounded treewidth. *The Computer Journal*, 51(3):255–269, 2008. (Page 118)

F. Bökler and P. Mutzel. Output-sensitive algorithms for enumerating the extreme nondominated points of multiobjective combinatorial optimization problems. In N. Bansal and I. Finocchi, editors, *Algorithms - ESA 2015: 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, pages 288–299. Springer, Berlin, 2015. (Pages 75, 76, 79)

S. Boussier, M. Vasquez, Y. Vimont, S. Hanafi, and P. Michelon. A multi-level search strategy for the 0–1 multidimensional knapsack problem. *Discrete Applied Mathematics*, 158(2):97–109, 2010. (Page 28)

V. Boyer, D. El Baz, and M. Elkihel. Solution of multidimensional knapsack problems via cooperation of dynamic programming and branch and bound. *European Journal of Industrial Engineering*, 4(4):434–449, 2010. (Page 28)

T. A. Brown and R. Strauch. Dynamic programming in multiplicative lattices. *Journal of Mathematical Analysis and Applications*, 2(12):364–370, 1965. (Page 24)

R. Buck. Partition of space. *The American Mathematical Monthly*, 50(9):541–544, 1943. (Page 36)

R. Burkard, E. Çela, P. Pardalos, and L. Pitsoulis. The quadratic assignment problem. *Handbook of Combinatorial Optimization*, 3, 1998. (Page 119)

A. Caprara, D. Pisinger, and P. Toth. Exact solution of the quadratic knapsack problem. *INFORMS Journal on Computing*, 11(2):125–137, 1999. (Pages 118, 132)

A. Caprara, H. Kellerer, U. Pferschy, and D. Pisinger. Approximation algorithms for knapsack problems with cardinality constraints. *European Journal of Operational Research*, 123(2):333–345, 2000. (Page 29)

M. E. Captivo, J. a. Clímaco, J. Figueira, E. Martins, and J. L. Santos. Soving bicriteria 0–1 knapsack problems using a labeling algorithm. *Computers & Operations Research*, 30(12):1865–1886, 2003. (Page 24)

A. Cerqueus. *Bi-Objective Branch-and-Cut Algorithms Applied to the Binary Knapsack Problem: Surrogate Upper Bound Sets, Dynamic Branching Strategies, Generation and Exploitation of Cover Inequalities*. Computer Science, Université de Nantes, 2015. (Page 29)

A. Cerqueus, A. Przybylski, and X. Gandibleux. Surrogate upper bound sets for bi-objective bi-dimensional binary knapsack problems. *European Journal of Operational Research*, 244(2):417–433, 2015. (Page 29)

V. Chankong and Y. Y. Haimes. *Multiobjective Decision Making: Theory and Methodology*. Elsevier Science Publishing Co., New York, 1983. (Pages 17, 93)

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT press, second edition, 2001. (Page 116)

K. Dächert. *Adaptive Parametric Scalarizations in Multicriteria Optimization*. Shaker Verlag, Aachen, 2014. (Pages 99, 113)

K. Dächert, K. Klamroth, R. Lacour, and D. Vanderpooten. Efficient computation of the search region in multi-objective optimization. *European Journal of Operational Research*, 2016. http://dx.doi.org/10.1016/j.ejor.2016.05.029. (Page 113)

G. B. Dantzig. Discrete-variable extremum problems. *Operations Research*, 5(2):266–277, 1957. (Page 22)

M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, Berlin Heidelberg, third edition, 2008. (Page 73)

C. Delort and O. Spanjaard. Using bound sets in multiobjective optimization: Application to the biobjective binary knapsack problem. In P. Festa, editor, *Experimental Algorithms: 9th International Symposium, SEA 2010, Ischia Island, Naples, Italy, May 20–22, 2010. Proceedings*, volume 6049 of *Lecture Notes in Computer Science*, pages 253–265. Springer, Berlin Heidelberg, 2010. (Page 25)

C. Delort and O. Spanjaard. A hybrid dynamic programming approach to the biobjective binary knapsack problem. *Journal of Experimental Algorithmics*, 18(2):1.2, 2013. (Page 25)

H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer, Berlin Heidelberg, 1987. (Pages 34, 35, 37, 40)

M. Ehrgott. *Multicriteria Optimization*. Springer, 2005. (Pages 13, 15, 31, 32, 33, 53)

M. Ehrgott and X. Gandibleux. A survey and annotated bibliography of multiobjective combinatorial optimization. *OR Spektrum*, 22:425–460, 2000. (Page 13)

M. Ehrgott, A. Löhne, and L. Shao. A dual variant of benson's "outer approximation algorithm" for multiple objective linear programming. *Journal of Global Optimization*, 52:757–778, 2012. (Page 75)

T. Erlebach, H. Kellerer, and U. Pferschy. Approximating multiobjective knapsack problems. *Management Science*, 48(12):1603–1612, 2002. (Pages 18, 26, 29)

S. L. Faulkenberg and M. M. Wiecek. On the quality of discrete representations in multiple objective programming. *Optimization and Engineering*, 11(3):423–440, 2010. (Page 18)

J.-A. Ferrez, K. Fukuda, and T. M. Liebling. Solving the fixed rank convex quadratic maximization in binary variables by a parallel zonotope construction algorithm. *European Journal of Operational Research*, 166:35–50, 2005. (Pages 37, 55)

J. R. Figueira, G. Tavares, and M. M. Wiecek. Labeling algorithms for multiple objective integer knapsack problems. *Computers & Operations Research*, 37(4):700–711, 2010. (Page 24)

J. R. Figueira, L. Paquete, M. Simões, and D. Vanderpooten. Algorithmic improvements on dynamic programming for the bi-objective {0,1} knapsack problem. *Computational Optimization and Applications*, 56(1):97–111, 2013. (Pages 24, 92, 95, 98, 103, 105)

J. R. Figueira, C. M. Fonseca, P. Halffmann, K. Klamroth, L. Paquete, S. Ruzika, B. Schulze, M. Stiglmayr, and D. Willems. Easy to say they are hard, but hard to see they are easy - towards a categorization of tractable multiobjective combinatorial optimization problems. *Journal of Multi-Criteria Decision Analysis*, 2016. http://dx.doi.org/10.1002/mcda.1574. (Pages 8, 13)

K. Florios, G. Mavrotas, and D. Diakoulaki. Solving multiobjective, multiconstraint knapsack problems using mathematical programming and evolutionary algorithms. *European Journal of Operational Research*, 203(1):14–21, 2010. (Page 29)

J. Forrest and T. Ralphs. Coin-OR Branch and Cut, March 2015. URL `https://projects.coin-or.org/Cbc`. (Page 103)

A. Fréville. The multidimensional 0–1 knapsack problem: An overview. *European Journal of Operational Research*, 155(1):1–21, 2004. (Page 27)

A. Fréville and G. Plateau. The 0–1 bidimensional knapsack problem: Toward an efficient high-level primitive tool. *Journal of Heuristics*, 2(2):147–167, 1996. (Page 28)

A. M. Frieze and M. R. B. Clarke. Approximation algorithms for the m-dimensional 0–1 knapsack problem: Worst-case and probabilistic analyses. *European Journal of Operational Research*, 15(1):100–109, 1984. (Page 29)

S. Gaas and T. Saaty. The computational algorithm for the parametric objective function. *Naval Research Logistics Quarterly*, 2:39–45, 1955. (Page 16)

G. Gallo, P. Hammer, and B. Simeone. Quadratic knapsack problems. In M. Padberg, editor, *Combinatorial Optimization*, volume 12 of *Mathematical Programming Studies*, pages 132–149. Springer, Berlin Heidelberg, 1980. (Pages 116, 118)

M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979. (Pages 13, 20, 117, 136)

B. Gavish and H. Pirkul. Efficient algorithms for solving multiconstraint zero-one knapsack problems to optimality. *Mathematical Programming*, 31(1):78–105, 1985. (Page 28)

G. V. Gens and E. V. Levner. Computational complexity of approximation algorithms for combinatorial problems. In *Proceedings of the 8th International Symposium on Mathematical Foundations of Computer Science*, volume 74 of *Lecture Notes in Computer Science*, pages 292–300. Springer, Moscow, 1979. (Page 29)

A. M. Geoffrion. Proper eficiency and the theory of vector maximization. *Journal of mathematical analysis and applications*, 22:618–630, 1968. (Page 16)

F. Glover. A multiphase-dual algorithm for the zero-one integer programming problem. *Operations Research*, 13(6):879–919, 1965. (Page 27)

F. Glover. Surrogate constraint duality in mathematical programming. *Operations Research*, 23(3):434–451, 1975. (Page 27)

C. Gomes da Silva and J. a. Clímaco. Using weighted-sum functions to compute nonsupported efficient solutions in multiobjective combinatorial-{0,1} problems. *International Journal of Information Technology & Decision Making*, 12(1):27–44, 2013. (Page 26)

C. Gomes da Silva, G. Clímaco, and J. R. Figueira. Geometrical configuration of the Pareto frontier of bi-criteria $\{0, 1\}$-knapsack problems. Research Report 16-2004, INESC-Coimbra, Portugal, 2004. (Page 31)

C. Gomes da Silva, J. a. Clímaco, and J. R. Figueira. New reduction strategy in the biobjective knapsack problem. *Computers & Operations Research*, 35(7):2292–2306, 2008. (Page 26)

J. Gorski. *Multiple Objective Optimization and Implications for Single Objective Optimization*. Shaker Verlag, Aachen, 2010. (Page 91)

J. Gorski, K. Klamroth, and S. Ruzika. Connectedness of efficient solutions in multiple objective combinatorial optimization. *Journal of Optimization Theory and Applications*, 150(3):475–497, 2011. (Page 32)

A. P. Guerreiro, C. M. Fonseca, and L. Paquete. Greedy hypervolume subset selection in low dimensions. *Evolutionary Computation*, 24(3):521–544, 2016. (Page 138)

Y. Y. Haimes, L. S. Lasdon, and D. A. Wismer. On a bicriterion formulation of the problems of integrated system identification and system optimization. *IEEE Transactions on Systems, Man, and Cybernetics*, 1(3):296–297, 1971. (Page 17)

H. Hamacher and K. Klamroth. *Lineare und Netzwerk-Optimierung*. Vieweg-Verlag, 2000. (Page 13)

S. Henn. Weight Constrained Minimum Spanning Tree Problems. Diploma thesis, Technische Universität Kaiserslautern, 2007. (Page 92)

F. Heyde and A. Löhne. Geometric duality in multiple objective linear programming. *SIAM Journal of Optimization*, 19(2):836–845, 2008. (Page 75)

E. L. Johnson, A. Mehrotra, and G. L. Nemhauser. Min-cut clustering. *Mathematical Programming*, 62(1):133–151, 1993. (Page 115)

N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984. (Page 37)

H. Kellerer and V. A. Strusevich. Fully polynomial approximation schemes for a symmetric quadratic knapsack problem and its scheduling applications. *Algorithmica*, 57(4):769–795, 2010. (Pages 118, 119)

H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, Heidelberg, 2004. (Pages 13, 20, 23)

K. Klamroth and J. Tind. Constrained optimization using multiple objective programming. *Journal of Global Optimization*, 37(3):325–355, 2007. (Page 91)

K. Klamroth and M. M. Wiecek. Dynamic programming approaches to the multiple criteria knapsack problem. *Naval Research Logistics*, 47:57–76, 2000. (Page 24)

K. Klamroth, R. Lacour, and D. Vanderpooten. On the representation of the search region in multi-objective optimization. *European Journal of Operational Research*, 245 (3):767–778, 2015. (Pages 99, 113)

P. J. Kolesar. A branch and bound algorithm for the knapsack problem. *Management Science*, 13(9):723–735, 1967. (Page 21)

M. Laumanns, L. Thiele, and E. Zitzler. An efficient, adaptive parameter variation scheme for metaheuristics based on the epsilon-constraint method. *European Journal of Operational Research*, 169(3):932–942, 2006. (Page 29)

A. Liefooghe, L. Paquete, and J. R. Figueira. On local search for bi-objective knapsack problems. *Evolutionary Computation*, 21(1):179–196, 2013. (Page 32)

J. H. Lorie and L. J. Savage. Three problems in rationing capital. *The Journal of Business*, 28(4):229–239, 1955. (Page 27)

L. Lozano and A. L. Medaglia. On an exact method for the constrained shortest path problem. *Computers & Operations Research*, 40(1):378–384, 2013. (Page 92)

T. Lust and J. Teghem. The multiobjective multidimensional knapsack problem: a survey and a new approach. *International Transactions in Operational Research*, 19(4):495–520, 2012. (Page 29)

R. Mansini and M. G. Speranza. Coral: An exact algorithm for the multidimensional knapsack problem. *INFORMS Journal on Computing*, 24(3):399–415, 2012. (Page 28)

H. M. Markowitz and A. S. Manne. On the solution of discrete programming problems. *Econometrica: Journal of the Econometric Society*, 25(1):84–110, 1957. (Page 27)

S. Martello and P. Toth. An upper bound for the zero-one knapsack problem and a branch and bound algorithm. *European Journal of Operational Research*, 1(3):169–175, 1977. (Pages 22, 98)

S. Martello and P. Toth. A new algorithm for the 0-1 knapsack problem. *Management Science*, 34(5):633–644, 1988. (Page 23)

S. Martello and P. Toth. *Knapsack Problems – Algorithms and Computer Implementations*. John Wiley & Sons, Inc., New York, 1990. (Pages 13, 25, 26)

S. Martello and P. Toth. An exact algorithm for the two-constraint 0–1 knapsack problem. *Operations Research*, 51(5):826–835, 2003. (Page 28)

S. Martello, D. Pisinger, and P. Toth. Dynamic programming and strong bounds for the 0–1 knapsack problem. *Management Science*, 45(3):414–424, 1999. (Page 23)

G. Mavrotas, J. R. Figueira, and K. Florios. Solving the bi-objective multi-dimensional knapsack problem exploiting the concept of core. *Applied Mathematics and Computation*, 215(7):2502–2514, 2009. (Page 29)

G. Mavrotas, J. R. Figueira, and A. Antoniadis. Using the idea of expanded core for the exact solution of bi-objective multi-dimensional knapsack problems. *Journal of Global Optimization*, 49(4):589–606, 2011. (Page 29)

G. Nemhauser and L. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, Inc., New York, USA, 1999. (Page 13)

G. L. Nemhauser and Z. Ullmann. Discrete dynamic programming and capital allocation. *Management Science*, 15(9):494–505, 1969. (Pages 21, 28)

Ö. Özpeynirci and M. Köksalan. An exact algorithm for finding extreme supported nondominated points of multiobjective mixed integer programs. *Management Science*, 56 (12):2302–2315, 2010. (Pages 74, 76, 79)

C. H. Papadimitriou and M. Yannakakis. On the approximability of trade-offs and optimal access of web sources. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, FOCS '00, pages 86–92, Washington, DC, USA, 2000. IEEE Computer Society. (Page 18)

U. Pferschy and J. Schauer. Approximation of the quadratic knapsack problem. *INFORMS Journal on Computing*, 28(2):308–318, 2016. (Page 118)

D. Pisinger. An expanding-core algorithm for the exact 0–1 knapsack problem. *European Journal of Operational Research*, 87(1):175–187, 1995. (Page 23)

D. Pisinger. A minimal algorithm for the 0–1 knapsack problem. *Operations Research*, 46 (5):758–767, 1997. (Pages 23, 86, 88, 103)

D. Pisinger. The quadratic knapsack problem - a survey. *Discrete Applied Mathematics*, 155(5):623–648, 2007. (Pages 117, 118)

D. Pisinger. Minknap algorithm, October 2015. URL `http://www.diku.dk/~pisinger/codes.html`. (Page 86)

D. Pisinger. Exact algorithm for the quadratic knapsack problem and instance generator, October 2016. URL `http://www.diku.dk/~pisinger/codes.html`. (Page 132)

D. W. Pisinger, A. B. Rasmussen, and R. Sandvik. Solution of large quadratic knapsack problems through aggressive reduction. *INFORMS Journal on Computing*, 19(2):280–290, 2007. (Page 118)

A. Przybylski, X. Gandibleux, and M. Ehrgott. A recursive algorithm for finding all nondominated extreme points in the outcome set of a multiobjective integer programme. *INFORMS Journal on Computing*, 22(3):371–386, 2010a. (Pages 35, 71, 73, 74, 79)

A. Przybylski, X. Gandibleux, and M. Ehrgott. A two phase method for multi-objective integer programming and its application to the assignment problem with three objectives. *Discrete Optimization*, 7(3):149–165, 2010b. (Page 76)

A. Przybylski, K. Klamroth, and R. Lacour. A simple and efficient dichotomic search algorithm for multi-objective integer linear programmes. Work in progress, 2017. (Pages 76, 79, 86, 88)

J. Puchinger, G. R. Raidl, and U. Pferschy. The multidimensional knapsack problem: Structure and algorithms. *INFORMS Journal on Computing*, 22(2):250–265, 2010. (Page 27)

D. J. Rader, Jr. and G. J. Woeginger. The quadratic 0–1 knapsack problem with series-parallel support. *Operations Research Letters*, 30(3):159–166, 2002. (Pages 118, 136)

J. M. W. Rhys. A selection problem of shared fixed costs and network flows. *Management Science*, 17(3):200–207, 1970. (Page 115)

C. D. Rodrigues, D. Quadri, P. Michelon, and S. Gueye. 0–1 quadratic knapsack problems: An exact approach based an a t-linearization. *SIAM Journal of Optimization*, 22(4): 1449–1468, 2012. (Page 118)

A. Rong and J. R. Figueira. A reduction dynamic programming algorithm for the bi-objective integer knapsack problem. *European Journal of Operational Research*, 231 (2):299–313, 2013. (Page 24)

A. Rong and J. R. Figueira. Dynamic programming algorithms for the bi-objective integer knapsack problem. *European Journal of Operational Research*, 236(1):85–99, 2014. (Page 24)

S. Ruzika. *On Multiple Objective Combinatorial Optimization*. Dr. Hut Verlag, München, 2008. (Page 92)

S. Ruzika and M. M. Wiecek. Approximation methods in multiobjective programming. *Journal of Optimization Theory and Applications*, 126(3):473–501, 2005. (Page 18)

H. M. Safer and J. B. Orlin. Fast approximation schemes for multi-criteria flow, knapsack, and scheduling problems. Technical Report 3757-95, Massachusetts Institute of Technology, 1995. (Page 26)

B. Schulze, L. Paquete, K. Klamroth, and J. R. Figueira. Bi-dimensional knapsack problems with one soft constraint. *Computers & Operations Research*, 78:15–26, 2017. (Page 8)

F. Seipp. *On Adjacency, Cardinality, and Partial Dominance in Discrete Multiple Objective Optimization*. Dr. Hut Verlag, München, 2013. (Pages 32, 89)

W. Shih. A branch and bound method for the multiconstraint zero-one knapsack problem. *The Journal of the Operational Research Society*, 30(4):369–378, 1979. (Page 28)

R. Taylor. Approximation of the quadratic knapsack problem. *Operations Research Letters*, 44(4):495–497, 2016. (Page 118)

A. Thesen. A recursive branch and bound algorithm for the multidimensional knapsack problem. *Naval Research Logistics Quarterly*, 22(2):341–353, 1975. (Page 28)

E. L. Ulungu and J. Teghem. Multi-objective combinatorial optimization problems: A survey. *Journal of Multi-Criteria Decision Analysis*, 3:83–104, 1994. (Page 13)

E. L. Ulungu and J. Teghem. Solving multi-objective knapsack problem by a branch-and-bound procedure. In J. a. Clímaco, editor, *Multicriteria Analysis: Proceedings of the XIth International Conference on MCDM, 1-6 August 1994, Coimbra, Portugal*, pages 269–278. Springer, Berlin Heidelberg, 1997. (Page 26)

D. Vanderpooten, L. Weerasena, and M. M. Wiecek. Covers and approximations in multiobjective optimization. *Journal of Global Optimization*, pages 1–19, 2016. (Page 18)

M. Visée, J. Teghem, M. Pirlot, and E. L. Ulungu. Two-phases method and branch and bound procedures to solve the bi-objective knapsack problem. *Journal of Global Optimization*, 12:139–155, 1998. (Pages 25, 31, 89)

H. M. Weingartner and D. N. Ness. Methods for the solution of the multidimensional 0/1 knapsack problem. *Operations Research*, 15(1):83–103, 1967. (Page 28)

C. Witzgall. Mathematical models of site selection of electronic message systems (EMS). Technical report, National Bureau of Standards, Washington, DC., 1975. (Page 115)

G. J. Woeginger. Sensitivity analysis for knapsack problems: another negative result. *Discrete Applied Mathematics*, 92:247–251, 1999. (Page 91)

L. A. Wolsey. *Integer Programming*. Series in discrete mathematics and optimization. Wiley-Interscience, 1998. (Page 34)

Z. Xu. A strongly polynomial FPTAS for the symmetric quadratic knapsack problem. *European Journal of Operational Research*, 218(2):377–381, 2012. (Page 119)

T. Zaslavsky. Facing up to arrangements: face-count formulas for partitions of space by hyperplanes. *Memoirs of the American Mathematical Society*, 1(154), 1975. (Pages 37, 38)

E. Zitzler and L. Thiele. Multiobjective optimization using evolutionary algorithms - a comparative case study. In A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature - PPSN V, 5th International Conference Amsterdam, The Netherlands, September 27-30, 1998, Proceedings*, volume 1498 of *Lecture Notes in Computer Science*, pages 292–301. Springer, Berlin Heidelberg, 1998. (Page 115)