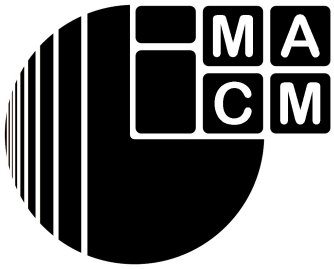

Bergische Universität Wuppertal

Fachbereich Mathematik und Naturwissenschaften

Institute of Mathematical Modelling, Analysis and Computational Mathematics (IMACM)

Preprint BUW-IMACM 17/17

B. Schulze, L. Paquete, K. Klamroth und J. R. Figueira

Bi-dimensional knapsack problems with one soft constraint

January 7, 2019

http://www.math.uni-wuppertal.de

Bi-dimensional knapsack problems with one soft constraint

Britta Schulze[a], Luís Paquete[b], Kathrin Klamroth[a], José Rui Figueira[c]

[a] *Department of Mathematics and Natural Sciences, University of Wuppertal, Germany*
[b] *CISUC, Department of Informatics Engineering, University of Coimbra, Portugal*
[c] *CEG-IST, Instituto Superior Técnico, Universidade de Lisboa, Portugal*

Abstract

In this article, we consider bi-dimensional knapsack problems with a soft constraint, 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 that 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.

Keywords: bi-dimensional knapsack problem, bi-objective knapsack problem, sensitivity analysis, soft constraints, dynamic programming

1. Introduction

Given a finite set of items with positive profits, weights, and a finite capacity, the 0–1-knapsack problem decides whether or not to include items, where each item can be included at most once. The goal is to maximize the overall profit of the included items under the constraint that the overall weight does not exceed the given capacity.

The knapsack problem (KP) is a classical problem in combinatorial optimization. It has applications in project selection, capital budgeting, and many others, and it appears as a frequent subproblem in more complex situations such as, for example, network design. Martello and Toth (1990) and Kellerer

Email addresses: schulze@math.uni-wuppertal.de (Britta Schulze), paquete@dei.uc.pt (Luís Paquete), klamroth@math.uni-wuppertal.de (Kathrin Klamroth), figueira@tecnico.ulisboa.pt (José Rui Figueira)

et al. (2004) give a detailed introduction to knapsack problems. The knapsack problem is NP-hard (see Garey and Johnson, 1979). It can be solved in polynomial time under smooth analysis using the dynamic programming (DP) approach by Nemhauser and Ullmann (1969) (see Beier and Vöcking, 2003, for more details). In practice, knapsack problems can be solved very efficiently by using, for instance, the core based algorithm by Pisinger (1997).

If one or several additional constraints are added to the formulation, a bi- or multi-dimensional knapsack problem is obtained. Such constraints could model, for example, different budget categories in capital budgeting applications. While the DP algorithms for the classical KP can be generalized also to multi-dimensional problems (see, for example, Nemhauser and Ullmann, 1969), they are generally much harder to solve in practice even in the bi-dimensional case. Gens and Levner (1979) showed that there exists no fully polynomial time approximation scheme for the bi-dimensional KP.

The multi-dimensional knapsack problem was first mentioned in the economical context of rationing capital (Lorie and Savage, 1955). Markowitz and Manne (1957) introduced formulations of discrete programming problems. They considered multi-dimensional knapsack problems among others and presented a general solution approach which can be adjusted to different discrete problem structures. 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). Weingartner and Ness (1967) and Nemhauser and Ullmann (1969) suggested dynamic programming as solution method. Balev et al. (2008) successfully combined upper bound computations using Linear Programming (LP) relaxations and a DP approach as an exact solution procedure. Gavish and Pirkul (1985) studied several relaxations and a reduction scheme. They also tested Branch & Bound (BB) based procedures with different branching, bounding, and separation techniques. Similar work was presented by Martello and Toth (2003) improving Lagrangian, surrogate, and LP-relaxations for the bi-dimensional KP. More recently, Boussier et al. (2010) applied different enumeration strategies that consider a given sequence of items w.r.t. the reduced costs of the non-basic variables in the corresponding LP-relaxations. They used Resolution Search, BB, and a simple Depth First Search (DFS) enumeration, depending on the level of the current branch. Mansini and Speranza (2012) presented a core algorithm for the multi-dimensional KP. They split the problem into subproblems with fewer variables and applied 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.

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 one-dimensional KPs, one may assume that *adjacent problems*, i.e., one-dimensional KPs where the right-hand sides differ by 1, have the same or at least similar optimal solutions. Blair (1998) showed that, even though this seems to be often the case, it cannot be expected in general. Woeginger (1999) proved a conjecture of Blair (1998) stating that already the decision problem asking whether the optimal solutions of two adjacent one-dimensional KPs share at least one selected item is NP-complete. Even worse so, Blair (1998) showed that for any pair of one-dimensional KPs, two adjacent one-dimensional KPs of larger size can be formulated that have the optimal solutions of the initial problems. Hence, all KPs with differing right-hand side values have to be solved individually in general.

We follow a different approach in this paper: 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 paper 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 was 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 was 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).

In this paper, 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. In the literature, bi-objective knapsack problems are usually studied with two maximization objectives (see, for example, Figueira et al., 2013, for exact solution methods). 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 paper is organized as follows. Section 2 gives the basic definitions for bi-dimensional and bi-objective knapsack problems, explains the transformation between the two problems and defines the dominance concepts. Section 3 presents the general structure of the DP algorithm. A preprocessing algorithm and the dominance relations that are applied during the DP are shown in Sec-

tion 4. Section 5 discusses the adaption of the DP algorithm for computing a predefined subset of the nondominated solutions. Finally, Section 6 reports computational experiments and the corresponding results. Conclusion and avenues for future research are presented in Section 7.

2. Bi-dimensional and bi-objective knapsack problems

We consider a bi-dimensional 0–1-knapsack problem (BDKP) which is formulated as follows:

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} p_i x_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} \bar{v}_i x_i \leqslant V \\ & \sum_{i=1}^{n} w_i x_i \leqslant W \\ & x_i \in \{0,1\} \qquad \forall i \in \{1,...,n\} \end{aligned} \tag{BDKP}$$

where $f_1(x) := \sum_{i=1}^{n} p_i x_i$ is the objective modeling, for example, the profit of a solution. The coefficient $p_i > 0$ indicates the profit of item i, $i \in \{1,...,n\}$. Two different capacity constraints are given with weight coefficients $\bar{v}_i > 0$ and $w_i > 0$, respectively, $i \in \{1,...,n\}$. The values V and W are the corresponding capacities of the knapsack. We assume that all coefficients $p_i,\ \bar{v}_i,\ w_i,\ i \in \{1,...,n\}$, and both capacities V and W belong to $\mathbb{N}$. Furthermore, to avoid trivial solutions, we assume that $\sum_{i=1}^{n} \bar{v}_i > V$, $\sum_{i=1}^{n} w_i > W$ and $\bar{v}_i < V,\ w_i < W,\ i \in \{1,...,n\}$. If item i is included in the knapsack, $x_i = 1$; otherwise $x_i = 0$, $i \in \{1,...,n\}$.

2.1. Reformulating constraints as objectives

Let us now assume that the second constraint with weight coefficients w_i and capacity W is a hard constraint, whereas the first constraint with weight coefficients $\bar{v}_i$ and capacity V is a soft constraint. Below we will 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 v_i such that $v_i := -\bar{v}_i < 0$, $i \in \{1,...,n\}$, we obtain a bi-objective knapsack problem (BOKP) with both maximization objectives:

$$\begin{aligned} \text{vmax} \quad & f(x) = \Big(\sum_{i=1}^{n} p_i x_i, \sum_{i=1}^{n} v_i x_i \Big) \\ \text{s.t.} \quad & \sum_{i=1}^{n} w_i x_i \leqslant W, \\ & x_i \in \{0,1\} \quad \forall i \in \{1,...,n\} \end{aligned} \tag{BOKP}$$

where $f_1(x) := \sum_{i=1}^{n} p_i x_i$ and $f_2(x) := \sum_{i=1}^{n} v_i x_i$ are the two objective functions. For convenience, we will refer to f_1 as the *original objective function* and to f_2 as the *transformed objective function*, respectively.

2.2. Dominance and related concepts

We denote the *set of feasible solutions* of (BOKP) as X and the *set of feasible points* in the objective space as $Y := f(X) = (f_1(X), f_2(X))$. A solution $x \in X$ is called *efficient* (or Pareto optimal) if there is no other solution $\bar{x} \in X$ such that

$$f_1(\bar{x}) \geqslant f_1(x), \quad f_2(\bar{x}) \geqslant f_2(x), \quad \text{with} \quad f(\bar{x}) \neq f(x).$$

The corresponding point $f(x)$ is called *nondominated*. $ND \subset Y$ denotes the *set of nondominated points*. If there exists a solution $\bar{x} \in X$ such that $f_1(\bar{x}) \geqslant f_1(x)$, $f_2(\bar{x}) \geqslant f_2(x)$ and $f(\bar{x}) \neq f(x)$, then we say x is *dominated by* $\bar{x}$ and $f(x)$ is *dominated by* $f(\bar{x})$.

Moreover, a solution $x \in X$ is called *weakly efficient* if and only if there exists no other solution $\bar{x} \in X$ such that

$$f_1(\bar{x}) > f_1(x) \quad \text{and} \quad f_2(\bar{x}) > f_2(x).$$

Let $\mathbb{R}^2_{\leqslant} := \{y \in \mathbb{R}^2 : y_1 \leqslant 0,\ y_2 \leqslant 0\}$ and let $Y_{\leqslant} := \text{conv}\{ND + \mathbb{R}^2_{\leqslant}\}$. The set of nondominated solutions ND can be partitioned into two disjoint subsets ND_u and ND_s as follows: On the one hand, nondominated points $f(x)$ which are located in the interior of $Y_{\leqslant}$ are denoted as *unsupported nondominated points*, and ND_u is the *set of unsupported nondominated points*. On the other hand, nondominated points $f(x)$ located on the boundary of $Y_{\leqslant}$ are called supported nondominated points or shortly *supported points*, the corresponding *set of supported points* is denoted by ND_s. The extreme points of $Y_{\leqslant}$ are called extreme supported nondominated points or *extreme points*. The *set of extreme points* ND_{es} is a subset of ND_s. In the following, we will use ND_{es} as an initial approximation for the set ND. We assume wlog that the extreme points $\bar{s}^1, ..., \bar{s}^m \in ND_{es}$ are sorted in increasing order of values of the original objective function.

2.3. Justification of the transformation

By applying the transformation from (BDKP) to (BOKP) we are interested in generating several alternative solutions to an optimal bi-dimensional solution $x^\star$ of (BDKP). In particular, $x^\star$ is an element of the set of weakly efficient solutions of (BOKP) and thus we do not loose any information by the transformation.

Indeed, from a multi-objective perspective (BDKP) can be seen as an ε-constraint scalarization of (BOKP), with f_2 reversely transformed into a constraint and the corresponding bound value ε set to V. Chankong and Haimes (1983) showed that every optimal solution of (BDKP) is weakly efficient for (BOKP), and at least one of the optimal solutions of (BDKP) is efficient for (BOKP). Therefore, if there is a unique optimal solution of (BDKP), it is an efficient solution of (BOKP) and the set of efficient solutions of (BOKP) contains at least one optimal solution of (BDKP).

In other words, if the nondominated set ND of (BOKP) is known, an optimal solution of (BDKP) is given by:

$$x^\star = \arg\max_{x \in X} \{f_1(x) \,:\, f(x) \in ND,\ f_2(x) \geqslant -V\}.$$

3. Dynamic programming algorithm

DP algorithms are based on implicit enumeration. In the case of the (BOKP), the procedure is split into n steps, denoted as *stages*. Each stage S_k, $k \in \{1, ..., n\}$, contains *states* $s = (s_1, s_2, s_3)$ corresponding to feasible solutions of problem (BOKP) and their images in the objective space. In particular, we assume that all states in a stage S_k, $k \in \{1, ..., n\}$, correspond to partial solutions $x \in \{0, 1\}^n$ in the sense that $x_{k+1} = ... = x_n = 0$. If a solution $x \in X$ 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.*, $s_3 = \sum_{i=1}^{n} w_i x_i$. In the following, we will use the notion of dominance also for states: A state s is dominated by another state $\bar{s}$ if and only if the corresponding solution x is dominated by $\bar{x}$. More precisely, s is dominated by $\bar{s}$ if and only if

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

Moreover, new stages S_k are created by adding (1-extension) or not adding (0-extension) the coefficients p_k, v_k and w_k 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 $\bar{s}_3 = s_3 + w_k$ is smaller than or equal to the capacity W, *i.e.*, if the corresponding solution stays feasible. In the following we will often analyze a state $s \in S_k$, $k \in \{1, ..., n\}$, and all of its extensions, hence we define the *set of feasible extensions of* s:

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

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

A central idea in dynamic programming is to make use of a principle of optimality: We are only interested in efficient solutions and, corresponding to that, in the nondominated set ND. 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. 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 ND 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 $k \in \{1, ..., n\}$, starting with $S_0 = \{(0, 0, 0)\}$:

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

Figure 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,

$k \in \{1, ..., n\}$, and a realized weight value w, $0 \leq w \leq W$. Therefore, several states can be allocated to one node, see Figure 1. Edges are connecting nodes of consecutive states, where a state allocated to a node in S_{k+1} has to be an extension of a state allocated to the node in S_k. We will thus use the term *DP network* in the following.

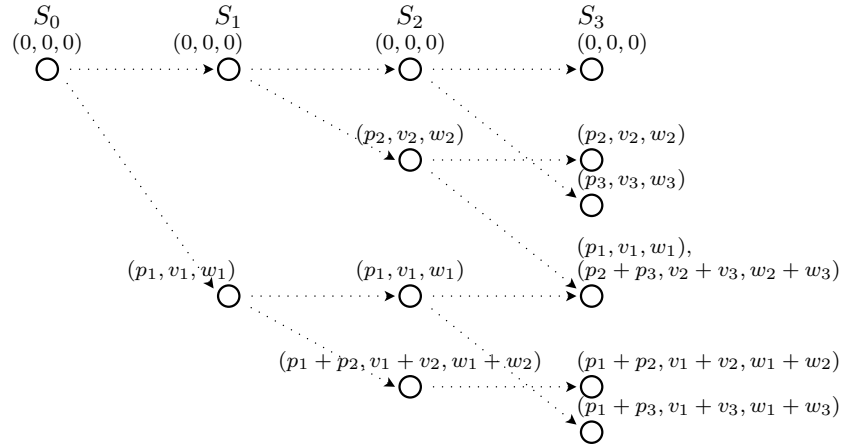

Figure 1: Example of Dynamic Programming network for problem (BOKP) where in this example we assume that $w_1 = w_2 + w_3$ and that $w_1 + w_2 + w_3 > W$

Algorithm 1 gives a pseudocode of the DP procedure. We use three different dominance relations (*DominanceA/B*, *UpperBound*, and *SearchZones*) specifically adapted to problem (BOKP) which will be discussed in detail in Section 4 below. Section 4.1 describes a preprocessing algorithm *Preprocessing* that computes ND_{es}. *Branch* generates the candidate set for the following stage on which the different dominance relations are performed.

Algorithm 1 Pseudocode of DP algorithm for problem (BOKP)

Input: n, $\mathcal{P} = \{p_1, ..., p_n\}$, $\mathcal{V} = \{v_1, ..., v_n\}$, $\mathcal{W} = \{w_1, ..., w_n\}$, W.
$ND_{es} :=$ *Preprocessing*($\mathcal{P}$, $\mathcal{V}$, $\mathcal{W}$, W)
$S := \{(0, 0, 0)\}$
for $k := 1, ..., n$ **do**
  $S :=$ *Branch*(S)
  if $k = n$ **then**
    $ND :=$ *DominanceB*(S)
  else
    $S :=$ *DominanceA*(S)
    $S :=$ *UpperBound*($\mathcal{P}$, $\mathcal{W}$, W, S, ND_{es})
    $S :=$ *SearchZones*($\mathcal{P}$, $\mathcal{V}$, $\mathcal{W}$, W, S, ND_{es})
  end if
end for
Output: *ND*

4. Preprocessing and pruning strategies

In this section we will describe the preprocessing and the pruning strategies of our DP algorithm. The set of extreme points ND_{es} has to be precomputed to apply the pruning strategies. Therefore, Section 4.1 first presents the *Preprocessing*. *DominanceA/B* and *UpperBound*, presented in Section 4.2, were introduced in Figueira et al. (2013) and are adapted here to the case of negative objective coefficients in the transformed objective f_2. The dominance relation *SearchZones*, presented in Section 4.3, uses a new idea based on these negative coefficients.

4.1. Preprocessing

The set of extreme points ND_{es} of problem (BOKP) can be computed using the dichotomic search by Aneja and Nair (1979); see Algorithm 2. First the two lexicographic maxima x_1 and x_2 are computed by solving a single-objective and one-dimensional knapsack problem where solely one of both objectives of (BOKP) is considered, respectively, and the constraint remains the same. All further nondominated points will lie within a *search zone* defined by the corresponding lexicographic points $f(x_1)$ and $f(x_2)$ as follows:

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

Note that $\mathcal{C}(f(x_1), f(x_2))$ defines a cone in $\mathbb{R}^2$. Continuing, the algorithm computes a weighted-sum objective defined by the values of $f(x_1)$ and $f(x_2)$. The optimal solution $\bar{x}$ of the resulting single-objective and one-dimensional knapsack problem corresponds to a supported point of (BOKP). It can be computed using well-known algorithms (see, for example, Kellerer et al., 2004).

The search zone $\mathcal{C}(f(x_1), f(x_2))$ can be split into two new search zones $\mathcal{C}(f(x_1), f(\bar{x}))$ and $\mathcal{C}(f(\bar{x}), f(x_2))$ 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. 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 point 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 2), the solution x_α, which corresponds to the upper left supported point defining the search zone, is included in the set E. This is sufficient for saving all found solutions since all solutions, except the lexicographic maximal solution x_2 which is saved beforehand, correspond to the upper left supported point for exactly one (discarded) search zone.

Note that supported but nonextreme solutions may not be detected. The reason is that the objective function value of the weighted-sum scalarization of a nonextreme solution $\bar{x}$ does not differ from that of the two solutions x_α and x_β which define $\mathcal{C}(f(x_\alpha), f(x_\beta))$ containing $f(\bar{x})$. Therefore, it is guaranteed that all extreme points are computed but no statement can be made about the nonextreme solutions.

To summarize, 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 inside $\mathcal{C}(f(x_\alpha), f(x_\beta))$ and it is deleted. For every search zone exactly one knapsack problem has to be solved. Since the number of supported points

is finite, the procedure stops after a finite number of iterations, computing the set $ND_{es} = \{\bar{s}^1, ..., \bar{s}^m\}$ and probably some additional nonextreme supported points.

Algorithm 2 Pseudocode of Dichotomic search

Input: $\mathcal{P} = \{p_1, ..., p_n\}$, $\mathcal{V} = \{v_1, ..., v_n\}$, $\mathcal{W} = \{w_1, ..., w_n\}$, W.
Compute lexicographic maximal solution x_2 with respect to $f_1(x)$
$f(x_2) := (f_1(x_2), f_2(x_2))$
Set $x_1 := (0, ..., 0)$, $f(x_1) := (0, 0)$ // *lex. max. solution w.r.t.* $f_2(x)$
$E := \{x_2\}$
if $f(x_1) \neq f(x_2)$ **then**
  $L := \{\mathcal{C}(f(x_1), f(x_2))\}$ // *list of search zones*
  $\ell := 1$
  while $\ell \geqslant 1$ **do**
    Select $\mathcal{C}(f(x_\alpha), f(x_\beta)) \in L$
    $\lambda_1 := f_1(x_\beta) - f_1(x_\alpha)$, $\lambda_2 := f_2(x_\alpha) - f_2(x_\beta)$
    $\bar{p}_i := \lambda_2 p_i + \lambda_1 v_i$, $i \in \{1, ..., n\}$
    Compute optimal solution $\bar{x}$ of the classical KP with profits $\bar{p}_i$ and weights w_i, $i \in \{1, ..., n\}$
    if $f(\bar{x}) = f(x_\alpha)$ **or** $f(\bar{x}) = f(x_\beta)$ **then**
      $E = E \cup \{x_\alpha\}$
      $L = L \setminus \{\mathcal{C}(f(x_\alpha), f(x_\beta))\}$
      $\ell = \ell - 1$
    else
      $L = L \cup \{\mathcal{C}(f(x_\alpha), f(\bar{x})), \mathcal{C}(f(\bar{x}), f(x_\beta))\} \setminus \{\mathcal{C}(f(x_\alpha), f(x_\beta))\}$
      $\ell = \ell + 1$
    end if
  end while
end if
Output: E // *E corresponds to U with $ND_{es} \subseteq U \subseteq ND_s$,* i.e.*, to a subset U of ND_s, which contains the complete set ND_{es}*

In our computational studies we used the code of Pisinger (1997) to solve the single-objective one-dimensional knapsack problem. 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 a subset of ND_{es}. It is also possible to start with an arbitrary approximation of ND_{es}, 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.

4.2. Dominance relations DominanceA/B *and* UpperBound

In Bazgan et al. (2009) three different dominance relations for (BOKP) 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**) will 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 $k+1, ..., n$ 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 $k, ..., n$, 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 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 $s_3 \geqslant \hat{s}_3$ 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 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.*, $e_1 \leqslant u_1(s)$ and $e_2 \leqslant u_2(s)$ for every $e \in \text{ext}(s)$. If $u(s)$ is already dominated by one of the extreme points $\bar{s} \in ND_{es}$, then s can be discarded because neither s itself nor one of the extensions will be nondominated. Let $s \in S_k$, $k \in \{1, ..., n\}$. The state s can be discarded, if there exists a point $\bar{s} \in ND_{es}$ with

$$\bar{s}_1 \geqslant u_1(s), \quad \bar{s}_2 \geqslant u_2(s) \qquad \text{and} \quad \bar{s}_1 \neq u_1(s) \text{ or } \bar{s}_2 \neq u_2(s).$$

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, 1990) for the classical knapsack problem. It only uses the coefficients of f_1 and the hard constraint. The not yet considered items $k+1, ..., n$ are ordered according to a non-increasing profit to weight ratio p_k/w_k. According to this order the items are added into the knapsack until the first item would violate the constraint. This item is called the critical item and is identified by the index c. The residual capacity $\overline{W}$ is calculated as follows:

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

To obtain an upper bound on f_1, the integrality constraint is relaxed for one item, but not for the critical item c, to get equality in the constraint. There are two possibilities for the optimal solution of this partially relaxed knapsack problem: The critical item is included or not. We consider all items $1, ..., c-1$ together with item c or $c+1$, respectively. Either item c is included, at the cost of removing a corresponding multiple $(w_c - \overline{W})/w_{c-1}$ of item $c-1$ (note that this multiple could be larger than 1), or the corresponding multiple $\overline{W}/w_{c+1}$ of

item $c+1$ is used to fill the remaining capacity. The maximum of both results is an upper bound on the first objective value. 2Additionally, the assumption that all data is integer allows to round this value down to the next integer:

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

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. 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 points in ND_{es}. 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): Let $\bar{s}^j$, $\bar{s}^{j+1}$ be two consecutive extreme points from the set ND_{es}, *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 $\mathcal{C}(\bar{s}^j, \bar{s}^{j+1}) = (\bar{s}_1^j, \bar{s}_2^{j+1}) + \mathbb{R}^2_{\geqslant}$, for all $j \in \{1, ..., m-1\}$. 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 2. In the following we will investigate the regions $[\bar{s}_1^j, \bar{s}_1^{j+1}) \times \mathbb{Z}^-$ for all $j \in \{1, ..., m-1\}$ and $\{\bar{s}_1^m\} \times \mathbb{Z}^-$ (illustrated for $j = 2$ in Figure 2), which also include the corresponding search triangles. To simplify the notation, we introduce a dummy point

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

and increase the last region to $[\bar{s}_1^m, \bar{s}_1^{m+1}) \times \mathbb{Z}^-$. For every region with $\bar{s}_1^{j+1} > s_1$ and $\bar{s}_1^j \leqslant u_1(s)$, $j \in \{1, ..., m\}$, we will 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 $\bar{s}_1^{j+1} \leqslant s_1$ are not of interest because $e_1 \geqslant s_1 \geqslant \bar{s}_1^{j+1}$ 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 $e_1 \leqslant u_1(s) < \bar{s}_1^j$ for all $e \in \text{ext}(s)$. The resulting procedure is named *SearchZones* in Algorithm 1. To simplify the notation, we partition the set of extensions of s into m subsets.

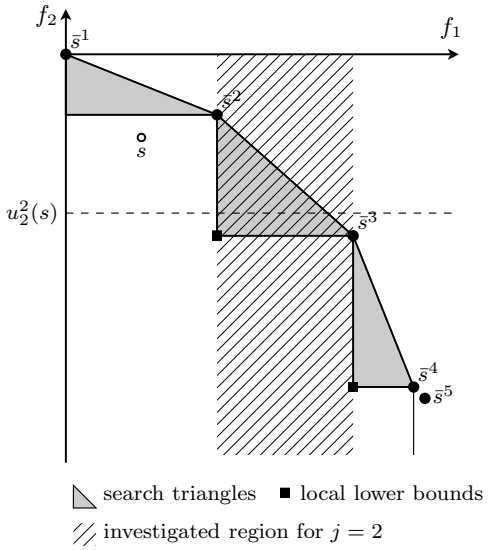

Figure 2: Illustration of search triangles, local lower bounds, the upper bound $u_2^2(s)$ for extensions e of state s with $\bar{s}_1^2 \leqslant e_1 < \bar{s}_1^3$, and the dummy point $\bar{s}^5$.

Definition 4.1. *Let $s \in S_k$, $k \in \{1, ..., n\}$. For every extreme point $\bar{s}^j \in ND_{es}$, $j \in \{1, ..., m\}$, let $\text{ext}^j(s)$ be a subset of $\text{ext}(s)$ with:*

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

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

i) It holds that

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

ii) Let $j \in \{1, ..., m\}$. If $\bar{s}_1^j > u_1(s)$ or if $\bar{s}_1^{j+1} \leqslant s_1$, 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.

Definition 4.3. *Let $s \in S_k$, $k \in \{1, ..., n\}$. Let $\sigma_1 : \{k+1, ..., n\} \to \{1, ..., n-k\}$, $i \mapsto \sigma_1(i)$ be a permutation that sorts the remaining items in non-increasing order of the coefficients p_i. For $\bar{s}^j \in ND_{es}$, $j \in \{1, ..., m\}$, with $s_1 < \bar{s}_1^{j+1}$ and $u_1(s) \geqslant \bar{s}_1^j$ let*

$$a_j = \min_b \left\{ b \in \{1, ..., n-k\} : s_1 + \sum_{\sigma_1(i)=1}^{b} p_{\sigma_1(i)} \geqslant \bar{s}_1^j \right\}.$$

*Now let $\sigma_2 : \{k+1, ..., n\} \to \{1, ..., n-k\}$, $i \mapsto \sigma_2(i)$ be a permutation, that sorts the remaining items in non-increasing order of the coefficients v_i (*i.e.,

in non-decreasing order w.r.t. the original, positive weight coefficients $\bar{v}_i$). For $j \in \{1, ..., m\}$, with $s_1 < \bar{s}_1^{j+1}$ and $u_1(s) \geqslant \bar{s}_1^j$, we define:

$$u_2^j(s) = s_2 + \sum_{\sigma_2(i)=1}^{a_j} v_{\sigma_2(i)}.$$

The value $u_2^j(s)$ is an upper bound on f_2 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 point $\bar{s}^j$. In particular, for all $j \in \{1, ..., m\}$ with $s_1 < \bar{s}_1^{j+1}$ and $u_1(s) \geqslant \bar{s}_1^j$, we have that $e_2 \leqslant u_2^j(s)$ for all $e \in \text{ext}^j(s)$. Now we can formulate the following theorem.

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

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

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 $e \notin ND$.

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

(2) It exists $j' \in \mathcal{J}$ such that $\widetilde{s} \in \text{ext}^{j'}(s)$ with $\bar{s}_1^{j'} \leqslant \widetilde{s}_1 < \bar{s}_1^{j'+1}$ and $\bar{s}_2^{j'} \geqslant \widetilde{s}_2 > \bar{s}_2^{j'+1}$, $\bar{s}^{j'}, \bar{s}^{j'+1} \in ND_{es} \cup \{\bar{s}^{m+1}\}$

We know that

- $\widetilde{s} \in \text{ext}(s) \ \Rightarrow \ \widetilde{s}_2 \leqslant u_2^{j'}(s)$
- $u_2^{j'}(s) \leqslant \bar{s}_2^{j'+1}$ because of (1)
- $\bar{s}_2^{j'+1} < \widetilde{s}_2$ because of (2)

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

If (1) holds for $s \in S_k$ with the corresponding set $\mathcal{J}$ (see Figure 3), the DP does not need to compute $\text{ext}(s)$. 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 ND.

If, for one or more values of $j \in \mathcal{J}$, (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 $j \in \{1, ..., m\}$. If condition (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 (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 $\widetilde{\mathcal{J}} = \left\{j \in \{1, ..., m\} \ : \ s_1 < \bar{s}_1^{j+1}\right\} \cap \left\{j \in \{1, ..., m\} \ : \ u_1(s) \geqslant \bar{s}_1^j\right\} \cap \mathcal{J}$.

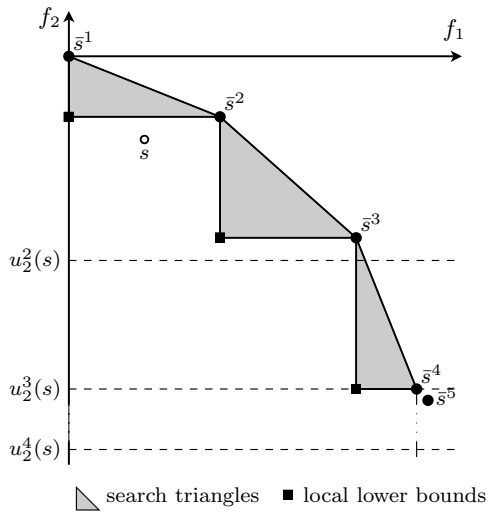

Figure 3: State s can be discarded: All upper bounds $u_2^j(s)$ are smaller than or equal to $\bar{s}_2^{j+1}$, $j \in \mathcal{J} = \{2, 3, 4\}$. Hence, all extensions $e \in \text{ext}(s)$, $e \neq s$, are dominated. In contrast, in Figure 2 the upper bound $u_2^2(s)$ is greater than $\bar{s}_2^3$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

5. Dynamic programming with cuts

In practice, it is unlikely that a decision-maker is interested in the whole set *ND*, 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 $[\bar{s}_1^{j_1}, \bar{s}_1^{j_2}] \times [\bar{s}_2^{j_2}, \bar{s}_2^{j_1}]$, $j_1, j_2 \in \{1, ..., m\}$, $j_1 < j_2$.

(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 $j = \max\{\hat{j} \in \{1, ..., m\} : \bar{s}_2^{\hat{j}} \geqslant -V\}$ and $\bar{s}^+ = \bar{s}^{j+1}$, *i.e.*, $\min\{\hat{j} \in \{1, ..., m\} : \bar{s}_2^{\hat{j}} < -V\} = j + 1$. 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:

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

- *DomR2*: States s in S_k could be discarded if $u_1(s) < \varepsilon_1$.
- *Bound*: (1) does not need to be checked for intervals with $\bar{s}_1^{j+1} < \varepsilon_1$ or $\bar{s}_2^j < \varepsilon_2$, $j \in \{1, ..., m\}$.

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

6. Computational results

The experiments were performed on an Intel Quadcore 2,80GHz with 4 GB RAM. The implementation of the DP algorithm was coded in C++. 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).

6.1. Experimental setup

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

Type A Profits p_i and weights $\bar{v}_i$ and w_i are integers uniformly generated in the range $[1, M]$, *i.e.*, profits $v_i \in [-M, -1]$, for all $i = 1, ..., n$.

Type B Profits p_i are integers uniformly generated in the range $[100, M - 100]$, weights $\bar{v}_i$ and w_i normal distributed with expectation $\mu = p_i$ 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 and v_i for (BOKP).

Type C Profits p_i are integers uniformly generated in the range $[100, M - 100]$, weights $\bar{v}_i$ and w_i are normal distributed with expectation $\mu = p_i$ and $\mu = M - p_i$, respectively, and standard deviation σ restricted to the range $[1, M - 1]$. This induces a positive correlation between profits p_i and weights $\bar{v}_i$, *i.e.*, a negative correlation between profits p_i and v_i for (BOKP), and a negative correlation between profits p_i and weights w_i.

Type D Profits p_i are integers uniformly generated in the range $[100, M - 100]$, weights $\bar{v}_i$ and w_i normal distributed with expectation $\mu = M - p_i$ and $\mu = p_i$, respectively, and standard deviation σ restricted to the range $[1, M - 1]$. This induces a negative correlation between profits p_i and weights $\bar{v}_i$, *i.e.*, a positive correlation between profits p_i and v_i for (BOKP), and a positive correlation between profits p_i and weights w_i.

Type E Profits p_i are integers uniformly generated in the range $[100, M - 100]$, weights $\bar{v}_i$ and w_i normal distributed with expectation $\mu = M - p_i$ and standard deviation σ restricted to the range $[1, M - 1]$. This induces a negative correlation between profits p_i and weights $\bar{v}_i$ and w_i, *i.e.*, a positive correlation between profits p_i and v_i for (BOKP).

The constraint slackness c_W is defined by $c_W \cdot \sum_{i=1}^{n} w_i = W$ for a constraint $\sum_{i=1}^{n} w_i \leqslant W$. Two values for the constraint slackness, namely $c_V = c_W = 0.25$ and $c_V = c_W = 0.75$, 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 $c_V = c_W = 0.25$ and $c_V = c_W = 0.75$ to test easy and hard instances for the DP algorithm, respectively.

The bounding induced by search triangles 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 tradeoff 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 5) were generated using the set of extreme points ND_{es}. The two supported points defining the search triangle that contains the optimal solution of (BDKP) (see Figure 4) shall be indicated by $\bar{s}^-$ and $\bar{s}^+$, with $\bar{s}_2^- \geqslant -V \geqslant \bar{s}_2^+$ (see again Section 5). The two lexicographic maxima are given by $\bar{s}^1$ and $\bar{s}^m$. 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 $\varepsilon_2 = \bar{s}_2^+ + R \cdot (\bar{s}_2^+ - \bar{s}_2^m)$. The regions of interest with $R = 1$, $R = 0.3$ and $R = 0$ are visualized for an exemplary problem instance in Figure 4. In particular, if $R = 0$, all nondominated points in the search triangle defined by $\bar{s}^-$ and $\bar{s}^+$ are computed.

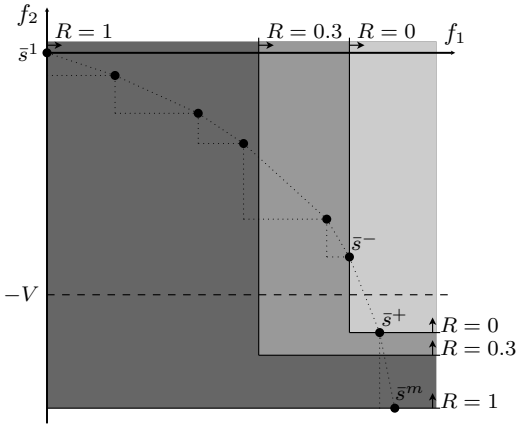

Figure 4: Illustration of regions of interest

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
	$\|ND\|$	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
	$\|ND\|$	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
	$\|ND\|$	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
	$\|ND\|$	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
	$\|ND\|$	1.30	68.50	110.30	155.90	195.70

Table 1: CPU-times of bi-objective and bi-dimensional approach in seconds and number of nondominated points for $n = 100$ items and $c_V = c_W = 0.25$.

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
	$\|ND\|$	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
	$\|ND\|$	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
	$\|ND\|$	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
	$\|ND\|$	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
	$\|ND\|$	1.50	139.00	231.00	295.70	357.20

Table 2: CPU-times of bi-objective and bi-dimensional approach in seconds and number of nondominated points for $n = 100$ items and $c_V = c_W = 0.75$.

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
	$\|ND\|$	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
	$\|ND\|$	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
	$\|ND\|$	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
	$\|ND\|$	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
	$\|ND\|$	2.70	14.60	24.70	61.20	117.50

Table 3: CPU-times of bi-objective and bi-dimensional approach in seconds and number of nondominated points for $n = 200$ items and $c_V = c_W = 0.25$.

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
	$\|ND\|$	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
	$\|ND\|$	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
	$\|ND\|$	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
	$\|ND\|$	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
	$\|ND\|$	1.10	51.50	101.00	225.10	390.90

Table 4: CPU-times of bi-objective and bi-dimensional approach in seconds and number of nondominated points for $n = 200$ items and $c_V = c_W = 0.75$.

6.2. Computation of the nondominated set

In the classical (BOKP) (with positive coefficients), every efficient solution is maximal in the sense that no further item can be included in the knapsack. 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 $|ND|$ and column $R = 1$ in Tables 1 and 2) is considerably higher than in classical BOKPs (see Figueira et al., 2013). As a consequence, the computational time for computing the whole set ND (rows DP and column $R = 1$ in Tables 1 and 2) is generally higher than for (BOKP), 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 milliseconds. All in all, the total computing times vary depending on the instance type, *i.e.*, the correlation structure and the constraint slackness.

6.3. Regions of interest

The CPU-times for several sizes of regions of interest are listed in the rows DP of Tables 1, 2, 3, and 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 5 and 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 $c_V = c_W = 0.25$, 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 $c_V = c_W = 0.75$, 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 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 $\sum_{i=1}^{n} v_i x_i \geqslant -V$ 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.

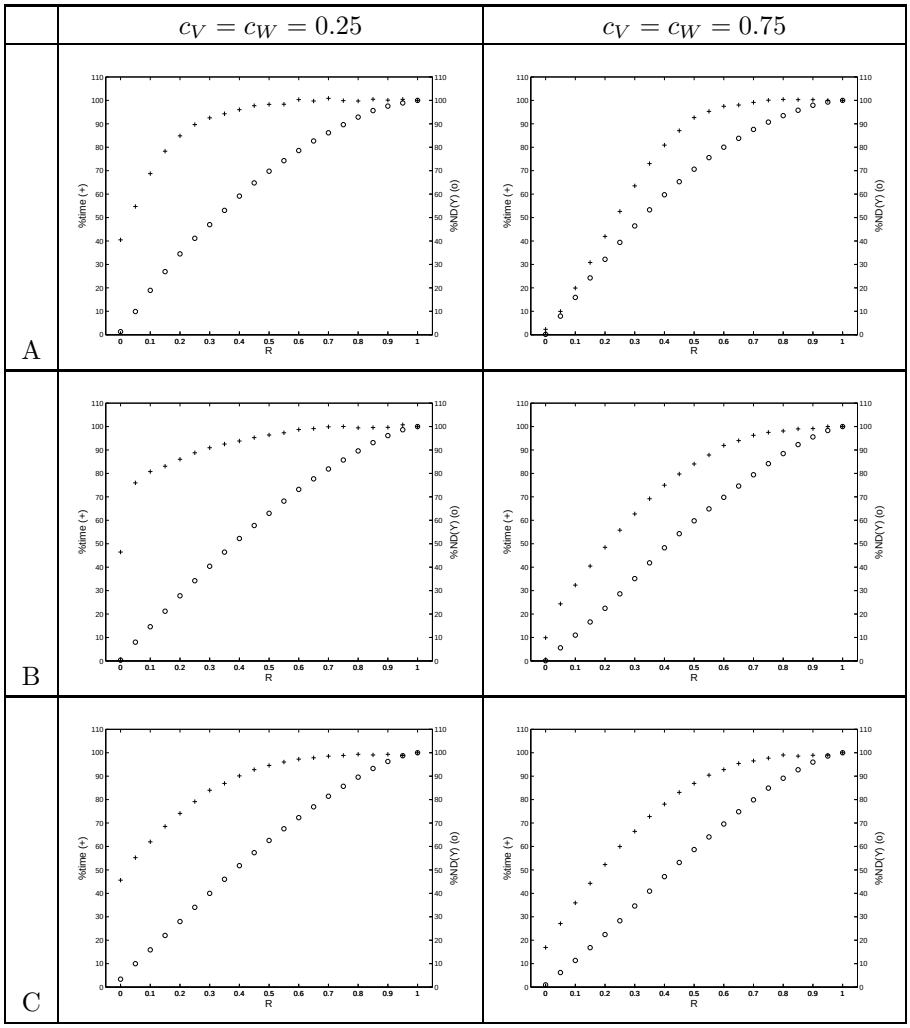

Figure 5: Computing times and number of nondominated points plotted for different values of R for instances of types A, B and C ($n = 100$). The case $R = 1$ is used as a reference (100%).

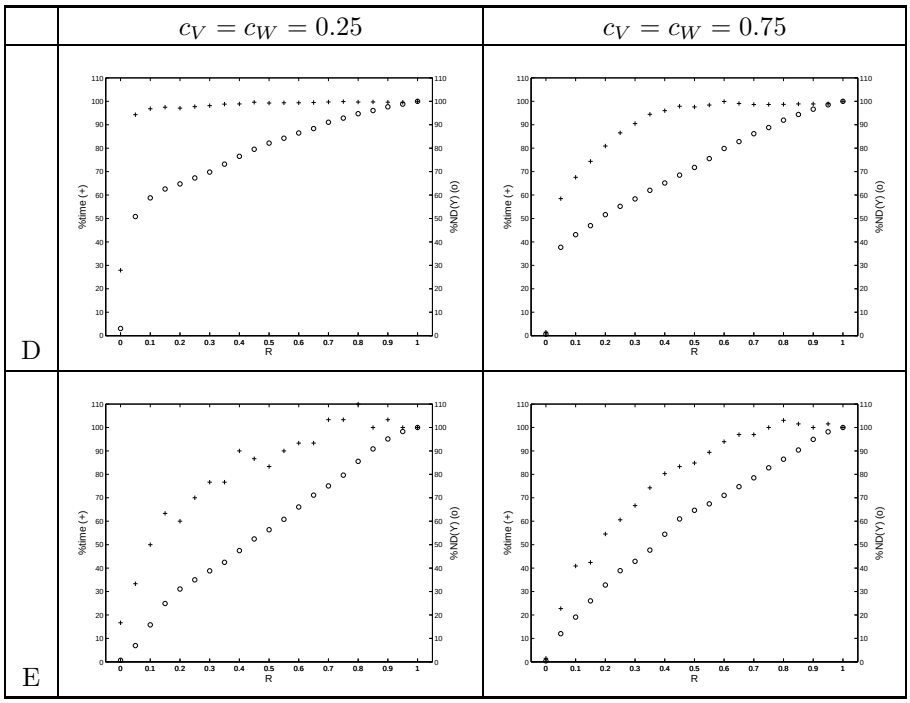

Figure 6: Computing times and number of nondominated points plotted for different values of R for instances of types D and E ($n = 100$). The case $R = 1$ is used as a reference (100%).

6.4. Comparison of (BDKP) *and* (BOKP)

We use the cbc-solver from the Coin-OR-library to compare the DP based solution approach to the direct solution of (BDKP). To get a fair comparison also in the case of search 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 V which, in turn, relates to applying the ε-constraint method (Haimes et al., 1971) to the bi-objective problem. First, we set $V = -\varepsilon_2$ and solve the associated knapsack problem. Afterwards, we use the previously computed optimal solution $\bar{x}$ and set $V = \sum_{i=1}^{n} \bar{v}_i \bar{x}_i - 1$ for the next instance. This continues until $f_1(\bar{x}) \leqslant \varepsilon_1$. The resulting CPU-times are presented in Tables 1 to 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 5 presents the average solution time for computing one solution of the bi-dimensional problem (BDKP), where all nondominated points, *i.e.*, $R = 1$, were computed for instances with 100 items and the nondominated points for $R = 0.01$ 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 5) and computing all solutions for $R = 0$ (Tables 1 to 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 5. Some particular instances are very hard to

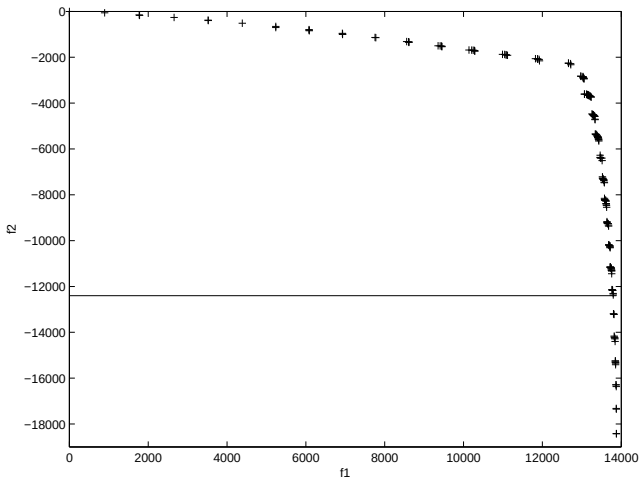

Figure 7: Set of nondominated points for one instance of type D.

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 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.

7. Conclusions

In this paper we presented a bi-objective 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 objective function with negative coefficients. Afterwards, we applied a bi-objective dynamic programming algorithm which uses special bounds induced by search zones. 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 can be defined, for example, by specifying ranges of acceptable and/or interesting levels of constraint satisfaction. Computational results indicate very good computing times in relation to the gained information.

		$c_V = c_W = 0.25$		$c_V = c_W = 0.75$	
	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	3399.41	5.37	181.80
E	t	0.24	8.69	0.46	1.72
	σ	0.55	308.12	1.17	6.17

Table 5: CPU-times (t in seconds) and standard deviation (σ) of cbc-solver solving (BDKP) 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).

An interesting question is whether an adapted preprocessing step for fixing variables would further reduce computing times. Especially in the case of small regions of interest, this seems to be a promising approach. Another direction of further research could be an extension of this approach to higher dimensions, for example, to tri-dimensional knapsack problems and to bi-dimensional, bi-objective knapsack problems.

Acknowledgments. This work was partially supported by the bilateral cooperation project Tractability in multiobjective combinatorial optimization funded by the Deutscher Akademischer Austausch Dienst (DAAD, Project-ID 57128839) and Fundação para a Ciência e Tecnologia (FCT), and by the European Regional Development Fund (FEDER), through the COMPETE 2020 Operational Program for Competitiveness and Internationalization (POCI).

References

Y. P. Aneja and K. P. K. Nair. Bicriteria transportation problem. *Management Science*, 25(1):73–78, 1979.

S. Balev, N. Yanev, A. Fréville, and R. Andonov. A dynamic programming based reduction procedure for the multidimensional 0–1 knapsack problem. *European Journal of Operational Research*, 186(1):63–76, 2008.

C. Bazgan, H. Hugot, and D. Vanderpooten. Solving efficiently the 0–1 multi-objective knapsack problem. *Computers & Operations Research*, 36(1):260–279, 2009.

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.

C. Blair. Sensitivity analysis for knapsack problems: a negative result. *Discrete Applied Mathematics*, 81:133–139, 1998.

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.

V. Chankong and Y. Y. Haimes. *Multiobjective Decision Making: Theory and Methodology*. Elsevier Science Publishing Co., New York, 1983.

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.

J. Forrest and T. Ralphs. Coin-OR Branch and Cut, March 2015. URL `https://projects.coin-or.org/Cbc`.

A. Fréville. The multidimensional 0–1 knapsack problem: An overview. *European Journal of Operational Research*, 155(1):1–21, 2004.

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.

B. Gavish and H. Pirkul. Efficient algorithms for solving multiconstraint zero-one knapsack problems to optimality. *Mathematical Programming*, 31(1): 78–105, 1985.

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.

J. Gorski. *Multiple Objective Optimization and Implications for Single Objective Optimization*. Shaker Verlag, Aachen, 2010.

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.

S. Henn. Weight Constrained Minimum Spanning Tree Problems. Diploma thesis, Technische Universität Kaiserslautern, 2007.

H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, Heidelberg, 2004.

K. Klamroth and J. Tind. Constrained optimization using multiple objective programming. *Journal of Global Optimization*, 37(3):325–355, 2007.

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.

J. H. Lorie and L. J. Savage. Three problems in rationing capital. *The Journal of Business*, 28(4):229–239, 1955.

L. Lozano and A. L. Medaglia. On an exact method for the constrained shortest path problem. *Computers & Operations Research*, 40(1):378–384, 2013.

R. Mansini and M. G. Speranza. Coral: An exact algorithm for the multidimensional knapsack problem. *INFORMS Journal on Computing*, 24(3):399–415, 2012.

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.

S. Martello and P. Toth. *Knapsack Problems Algorithms and Computer Implementations*. John Wiley & Sons, New York, 1990.

S. Martello and P. Toth. An exact algorithm for the two-constraint 0–1 knapsack problem. *Operations Research*, 51(5):826–835, 2003.

G. L. Nemhauser and Z. Ullmann. Discrete dynamic programming and capital allocation. *Management Science*, 15(9):494–505, 1969.

D. Pisinger. A minimal algorithm for the 0–1 knapsack problem. *Operations Research*, 46(5):758–767, 1997.

J. Puchinger, G. R. Raidl, and U. Pferschy. The multidimensional knapsack problem: Structure and algorithms. *INFORMS Journal on Computing*, 22 (2):250–265, 2010.

S. Ruzika. *On Multiple Objective Combinatorial Optimization*. Dr. Hut Verlag, München, 2008.

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.

G. J. Woeginger. Sensitivity analysis for knapsack problems: another negative result. *Discrete Applied Mathematics*, 92:247–251, 1999.