Solving efficiently the 0-1 multi-objective knapsack problem

Cristina BAZGAN, Hadrien HUGOT*, and Daniel VANDERPOOTEN

LAMSADE, Université Paris-Dauphine,

Place du Maréchal de Lattre de Tassigny, 75 775 Paris Cedex 16, France

Fax.: +33 1 44 05 40 91

`{bazgan,hugot,vdp}@lamsade.dauphine.fr`

* corresponding author

Abstract

In this paper, we present an approach, based on dynamic programming, for solving the 0-1 multi-objective knapsack problem. The main idea of the approach relies on the use of several complementary dominance relations to discard partial solutions that cannot lead to new non-dominated criterion vectors. This way, we obtain an efficient method that outperforms the existing methods both in terms of CPU time and size of solved instances.

Extensive numerical experiments on various types of instances are reported. A comparison with other exact methods is also performed. In addition, for the first time to our knowledge, we present experiments in the three-objective case.

Keywords: multi-objective knapsack problem, non-dominated criterion vectors, efficient solutions, dynamic programming, dominance relations, combinatorial optimization.

1 Introduction

In multi-objective combinatorial optimization, a major challenge is to develop efficient procedures to generate efficient solutions, that have the property that no improvement on any objective is possible without sacrificing on at least another objective. The aim is thus to find the efficient set (which consists of all the efficient solutions) or, more often, a reduced efficient set (which consists of only one solution for each non-dominated criterion vector). A survey and an annotated bibliography about multi-objective combinatorial optimization can be found in [1] and [2].

This paper deals with a particular multi-objective combinatorial optimization problem: the 0-1 multi-objective knapsack problem. The single-objective version of this problem has been studied extensively in the literature (see, *e.g.*, [3, 4]). Moreover, in the multi-objective case, many real-world applications are reported dealing with capital budgeting [5], selection of transportation investment alternatives [6], relocation issues arising in conservation biology [7], and planning remediation of contaminated lightstation sites [8].

Several exact approaches have been proposed in the literature to find the efficient set or a reduced efficient set for the multi-objective knapsack problem. We first mention a theoretical work [9], without experimental results, where several dynamic programming formulations are presented. Two specific methods, with extensive experimental results, have been proposed: the two-phase method including a branch and bound algorithm proposed in [10], and the method of Captivo *et al.* presented in [11], based on a transformation of the problem into a bi-objective shortest path problem which is solved using a labeling algorithm. We can also mention the recent work of Silva *et al.* [12]. All these methods have been especially designed for the bi-objective case. Besides exact methods investigated in this paper, approximation algorithms [13] and metaheuristics [14, 15, 16] have been proposed.

In this paper, we present a new approach based on dynamic programming. The main idea of the approach relies on the use of several complementary dominance relations to discard partial solutions that cannot lead to new non-dominated criterion vectors. This way, we obtain an efficient method that outperforms the existing methods both in terms of CPU time and size of solved instances (up to 4000 items in less than 2 hours in the bi-objective case). In our experiments, we compare our approach with the method proposed in [11], which is the most efficient method currently known, and with an exact method based on a commercial Integer Programming solver. In addition, for the first time to our knowledge, we present experiments in the three-objective case.

This paper is organized as follows. In section 2, we review basic concepts about multi-objective optimization and formally define the multi-objective knapsack problem. Section 3 presents and establishes the validity of a dynamic programming approach based on several dominance relations. Section 4 is devoted to implementation issues. Computational experiments and results are reported in section 5. Conclusions are provided in a final section.

2 Preliminaries

2.1 Multi-objective optimization

Consider a multi-objective optimization problem with p criteria or objectives where X denotes the finite set of feasible solutions. Each solution $x \in X$ is represented in the criterion space by its corresponding criterion vector $f(x) = (f_1(x), \dots, f_p(x))$. We assume in the following that each criterion has to be maximized.

From these p criteria, the dominance relation defined on X, denoted by $\underline{\Delta}$, states that a feasible solution x dominates a feasible solution x', $x\underline{\Delta}x'$, if and only if $f_i(x) \geq f_i(x')$ for $i = 1, \dots, p$. We denote by Δ the asymmetric part of $\underline{\Delta}$. A solution x is *efficient* if and only if there is no other feasible solution $x' \in X$ such that $x'\Delta\ x$, and its corresponding criterion vector is said to be *non-dominated.* Thus, the *efficient set* is defined as $E(X) = \{x \in X : \forall x' \in X,\ \text{not}(x'\Delta x)\}$. The set of non-dominated criterion vectors, which corresponds to the image of the efficient set in the criterion space, is denoted by ND. Since the efficient set can contain different solutions corresponding to the same criterion vector, any subset of $E(X)$ that contains one and only one solution for every non-dominated criterion vector is called a *reduced efficient set.* Observe that $X' \subseteq X$ is a reduced efficient set if and only if it is a *covering* and *independent set* of X with respect to $\underline{\Delta}$. We recall that, given $\succsim$ a binary relation defined on a finite set A,

- $B \subseteq A$ is a *covering (or dominating) set* of A with respect to $\succsim$ if and only if for all $a \in A \backslash B$ there exists $b \in B$ such that $b \succsim a$,
- $B \subseteq A$ is an *independent (or stable) set* with respect to $\succsim$ if and only if for all $b, b' \in B,\ b \neq b',\ \text{not}(b \succsim b')$.

2.2 The $0-1$ multi-objective knapsack problem

An instance of the $0-1$ multi-objective knapsack problem consists of an integer capacity $W > 0$ and n items. Each item k has a positive integer weight w^k and p non negative integer profits $v_1^k, \dots, v_p^k$ $(k = 1, \dots, n)$. A feasible solution is represented by a vector $x = (x_1, \dots, x_n)$ of binary decision variables x_k, such that $x_k = 1$ if item k is included in the solution and 0 otherwise, which satisfies the weight constraint $\sum_{k=1}^n w^k x_k \leq W$. The value of a feasible solution $x \in X$ on the ith objective is $f_i(x) = \sum_{k=1}^n v_i^k x_k$ $(i = 1, \dots, p)$. For any instance of this problem, we aim at determining the set of non-dominated criterion vectors.

3 Dynamic Programming and dominance relations

We first describe the sequential process used in Dynamic Programming (DP) and introduce some basic concepts of DP (section 3.1). Then, we present the concept of multiple dominance relations in DP (section 3.2). Section 3.3 indicates a manner to use efficiently a dominance relation.

3.1 Sequential process and basic concepts of DP

The sequential process used in DP consists of n phases. At any phase k we generate the set of states S^k which represents all the feasible solutions made up of items belonging exclusively to the k first items ($k = 1, \ldots, n$). A state $s^k = (s^k_1, \ldots, s^k_p, s^k_{p+1}) \in S^k$ represents a feasible solution of value s^k_i on the ith objective ($i = 1, \ldots, p$) and of weight s^k_{p+1}. Thus, we have $S^k = S^{k-1} \cup \{(s^{k-1}_1 + v^k_1, \ldots, s^{k-1}_p + v^k_p, s^{k-1}_{p+1} + w^k) : s^{k-1}_{p+1} + w^k \leq W, s^{k-1} \in S^{k-1}\}$ for $k = 1, \ldots, n$ where the initial set of states S^0 contains only the state $s^0 = (0, \ldots, 0)$ corresponding to the empty knapsack. In the following, we identify a state and its corresponding feasible solution. As a consequence, relation $\underline{\Delta}$ defined on X is also valid on S^k, and we have $s^k \underline{\Delta} \tilde{s}^k$ if and only if $s^k_i \geq \tilde{s}^k_i$, $i = 1, \ldots, p$.

Definition 1 (Completion, extension, restriction) *For any state $s^k \in S^k$ ($k < n$), a* completion *of s^k is any, possibly empty, subset $J \subseteq \{k+1, \ldots, n\}$ such that $s^k_{p+1} + \sum_{j \in J} w^j \leq W$. We assume that any state $s^n \in S^n$ admits the empty set as unique completion. A state $s^n \in S^n$ is an* extension *of $s^k \in S^k$ ($k \leq n$) if and only if there exists a completion J of s^k such that $s^n_i = s^k_i + \sum_{j \in J} v^j_i$ for $i = 1, \ldots, p$ and $s^n_{p+1} = s^k_{p+1} + \sum_{j \in J} w^j$. The set of extensions of s^k is denoted by $Ext(s^k)$ ($k \leq n$). Finally, $s^k \in S^k$ ($k \leq n$) is a* restriction *at phase k of state $s^n \in S^n$ if and only if s^n is an extension of s^k.*

3.2 Dominance relations in Dynamic Programming

The efficiency of DP depends crucially on the possibility of reducing the set of states at each phase. For this purpose, dominance relations between states are used to discard states at any phase. A dominance relation is defined as follows.

Definition 2 (Dominance relation between states) *A relation D^k on S^k, $k = 1, \ldots, n$, is a dominance relation, if for all $s^k, \tilde{s}^k \in S^k$,*

$$s^k D^k \tilde{s}^k \Rightarrow \forall \tilde{s}^n \in Ext(\tilde{s}^k), \exists s^n \in Ext(s^k), s^n \underline{\Delta} \tilde{s}^n \tag{1}$$

Although dominance relations are not transitive by definition, they are usually transitive by construction. This is the case, indeed, with the three relations used in our implementation (see Section 4.2). Observe also that if D^k is a dominance relation then its transitive closure $\widehat{D}^k$ is a dominance relation. Finally, if D^k_i, $i = 1, \ldots, m$, are dominance relations then $D^k = \bigcup_{i=1}^m D^k_i$ is also a dominance relation, which is generally non-transitive even if relations D^k_i are transitive.

In an efficient implementation of DP, it is desirable to make use of multiple dominance relations $D^k_1, \ldots, D^k_m$ ($m \geq 1$) at phase k ($k = 1, \ldots, n$) since each dominance relation D^k_i

($i = 1, \dots, m$) focuses on specific considerations. We introduce now a way of using multiple dominance relations in Algorithm 1. At each phase k, Algorithm 1 generates a subset of states $C^k \subseteq S^k$. This is achieved by first creating from C^{k-1} a temporary subset $C_0^k \subseteq S^k$. Then, we apply dominance relations $D_1^k, \dots, D_m^k$ sequentially. This is done by retaining for $i = 1, \dots, m$, C_i^k which can be *any* covering set of C_{i-1}^k with respect to D_i^k.

Algorithm 1: Dynamic Programming with multiple dominance relations

```
1 $C^0 \leftarrow \{(0, \dots, 0)\}$;
2 for $k \leftarrow 1$ to $n$ do
3     $C_0^k \leftarrow C^{k-1} \cup \{(s_1^{k-1} + v_1^k, \dots, s_p^{k-1} + v_p^k, s_{p+1}^{k-1} + w^k) | s_{p+1}^{k-1} + w^k \leq W : s^{k-1} \in C^{k-1}\}$;
4     for $i \leftarrow 1$ to $m$ do  determine $C_i^k$ any covering set of $C_{i-1}^k$ with respect to $D_i^k$;
5     $C^k \leftarrow C_m^k$;
6 return $C^n$;
```

The following result characterizes the set C_m^k obtained at the end of each phase k.

Proposition 1 *For any dominance relations $D_1^k, \dots, D_m^k$ ($m \geq 1$) on S^k, the set C_m^k obtained by Algorithm 1 at each phase is a covering set of C_0^k with respect to $D^k = \widehat{\bigcup_{i=1}^m D_i^k}$ ($k = 1, \dots, n$).*

Proof: Considering $s^k \in C_0^k \backslash C_m^k$, it has been removed when selecting a covering set at an iteration of step 4. Let $i_1 \in \{1, \dots, m\}$ be the iteration of step 4 such that $s^k \in C_{i_1-1}^k \backslash C_{i_1}^k$. Since $C_{i_1}^k$ is a covering set of $C_{i_1-1}^k$ with respect to $D_{i_1}^k$, there exists $\tilde{s}_{(1)}^k \in C_{i_1}^k$ such that $\tilde{s}_{(1)}^k D_{i_1}^k s^k$. If $\tilde{s}_{(1)}^k \in C_m^k$ then the covering property holds, since $D_{i_1}^k \subseteq D^k$. Otherwise, there exists an iteration $i_2 > i_1$, corresponding to the iteration of step 4 such that $\tilde{s}_{(1)}^k \in C_{i_2-1}^k \backslash C_{i_2}^k$. As before, we establish that there exists $\tilde{s}_{(2)}^k \in C_{i_2}^k$ such that $\tilde{s}_{(2)}^k D_{i_2}^k \tilde{s}_{(1)}^k$. Since $D_{i_2}^k \subseteq D^k$, we get that $\tilde{s}_{(2)}^k D^k \tilde{s}_{(1)}^k D^k s^k$ and by transitivity of D^k, we ensure that $\tilde{s}_{(2)}^k D^k s^k$. By repeating this process, we establish the existence of a state $\tilde{s}^k \in C_m^k$, such that $\tilde{s}^k D^k s^k$. □

We give now conditions under which Algorithm 1 generates the set *ND* of non-dominated criterion vectors.

Theorem 1 *For any family of dominance relations D_i^k ($i = 1, \dots, m; k = 1, \dots, n$), Algorithm 1 returns C^n which is a covering set of S^n with respect to $\underline{\Delta}$. Moreover, if at phase n we use at least one relation $D_i^n = \underline{\Delta}$ and impose that the selected covering set C_i^n is also independent with respect to D_i^n then C^n represents the set ND of non-dominated criterion vectors.*

Proof: Considering $\tilde{s}^n \in S^n \backslash C^n$, all its restrictions have been removed when retaining a covering set with respect to $D^k = \widehat{\bigcup_{i=1}^m D_i^k}$ during phases $k \leq n$. Let k_1 be the highest phase where $C_0^{k_1}$ still contains restrictions of $\tilde{s}^n$, which will be removed by applying one of the relations $D_i^{k_1}$ $(i = 1, \ldots, m)$. Consider any of these restrictions, denoted by $\tilde{s}_{(n)}^{k_1}$. Since $\tilde{s}_{(n)}^{k_1} \in C_0^{k_1} \backslash C^{k_1}$, we know from Proposition 1, that there exists $s^{k_1} \in C^{k_1}$ such that $s^{k_1} D^{k_1} \tilde{s}_{(n)}^{k_1}$. By (1), since D^k is a dominance relation, we have that for all extensions of $\tilde{s}_{(n)}^{k_1}$, and in particular for $\tilde{s}^n$, there exists $s^{n_1} \in \mathrm{Ext}(s^{k_1})$ such that $s^{n_1} \underline{\Delta} \tilde{s}^n$. If $s^{n_1} \in C^n$, then the covering property holds. Otherwise, there exists a phase $k_2 > k_1$, corresponding to the highest phase where $C_0^{k_2}$ still contains restrictions of s^{n_1}, which will be removed by applying one of the relation $D_i^{k_2}$ $(i = 1, \ldots, m)$. Consider any of these restrictions, denoted by $s_{(n_1)}^{k_2}$. As before, we establish the existence of a state $s^{k_2} \in C^{k_2}$ such that there exists $s^{n_2} \in \mathrm{Ext}(s^{k_2})$ such that $s^{n_2} \underline{\Delta} s^{n_1}$. Transitivity of $\underline{\Delta}$ ensures that $s^{n_2} \underline{\Delta} \tilde{s}^n$. By repeating this process, we establish the existence of a state $s^n \in C^n$, such that $s^n \underline{\Delta} \tilde{s}^n$.

In addition, by selecting a set C_i^n that is independent with respect to $D_i^n = \underline{\Delta}$, this property remains valid for C_m^n which is a subset of C_i^n. Thus C^n, which corresponds to a reduced efficient set, represents the set of non-dominated vectors. $\square$

The previous theorem only requires that one of the $n.m$ covering sets is independent with respect to its corresponding dominance relation. Even if all other sets C_i^k can be *any* covering sets, practical efficiency of Algorithm 1 induces to select covering sets of minimal size.

This can be easily achieved when dominance relations D_i^k are transitive, by selecting, at step 4 of Algorithm 1, covering sets C_i^k that are independent with respect to D_i^k. It is well-know indeed that a covering and independent set (*i.e.* a kernel) with respect to a transitive relation does exist and is a covering set of minimal size (see, *e.g.*, [17]).

3.3 Generating covering and independent sets

We present now in Algorithm 2 a way of of producing C_i^k a covering and independent set of C_{i-1}^k with respect to a transitive relation D_i^k (step 4 of Algorithm 1).

Proposition 2 *For any transitive dominance relation D_i^k on S^k, Algorithm 2 returns C_i^k a covering and independent set of C_{i-1}^k with respect to D_i^k $(k = 1, \ldots, n; i = 1, \ldots, m)$.*

Proof: Clearly, C_i^k is independent with respect to D_i^k, since we insert a state s^k into C_i^k at step 12 only if it is not dominated by any other state of C_i^k (step 5) and all states dominated by s^k have been removed from C_i^k (steps 6 and 10).

We show now that C_i^k is a covering set of C_{i-1}^k with respect to D_i^k. Consider $\tilde{s}^k \in C_{i-1}^k \backslash C_i^k$. This occurs either because it did not pass the test at step 5 or was removed at step 6 or 10. This is due respectively to a state $\bar{s}^k$ already in C_i^k or to be included in C_i^k (at step 12) such that $\bar{s}^k D_i^k \tilde{s}^k$. It may happen that $\bar{s}^k$ will be removed from C_i^k at a later iteration of the **for** loop (at step 6 or 10) if there exists a new state $\hat{s}^k \in C_{i-1}^k$ to be included in C_i^k, such that $\hat{s}^k D_i^k \bar{s}^k$. However, transitivity of D_i^k ensures the existence, at the end of phase k, of a state $s^k \in C_i^k$ such that $s^k D_i^k \tilde{s}^k$. $\square$

Algorithm 2: Compute C_i^k a covering and independent set of C_{i-1}^k with respect to a transitive relation D_i^k

```
   /* Assume that $C_{i-1}^k = \{s^{k(1)}, \dots, s^{k(r)}\}$ */
1  $C_i^k \leftarrow \{s^{k(1)}\}$;
2  for $h \leftarrow 2$ to $r$ do
       /* Assume that $C_i^k = \{\tilde{s}^{k(1)}, \dots, \tilde{s}^{k(\ell_h)}\}$ */
3      $dominated \leftarrow$ false ; $dominates \leftarrow$ false ; $j \leftarrow 1$;
4      while $j \leq \ell_h$ and not(dominated) and not(dominates) do
5          if $\tilde{s}^{k(j)} D_i^k s^{k(h)}$ then $dominated \leftarrow$ true
6          else if $s^{k(h)} D_i^k \tilde{s}^{k(j)}$ then $C_i^k \leftarrow C_i^k \backslash \{\tilde{s}^{k(j)}\}$ ; $dominates \leftarrow$ true;
7          $j \leftarrow j + 1$;
8      if not(dominated) then
9          while $j \leq \ell_h$ do
10             if $s^{k(h)} D_i^k \tilde{s}^{k(j)}$ then $C_i^k \leftarrow C_i^k \backslash \{\tilde{s}^{k(j)}\}$;
11             $j \leftarrow j + 1$;
12         $C_i^k \leftarrow C_i^k \cup \{s^{k(h)}\}$;
13 return $C^k$;
```

Algorithm 2 can be improved since it is usually possible to generate states of $C_{i-1}^k = \{s^{k(1)}, \dots, s^{k(r)}\}$ according to a *dominance preserving order* for D_i^k such that for all $\ell < j$ $(1 \leq \ell, j \leq r)$ we have either $s^{k(\ell)} D_i^k s^{k(j)}$ or not$(s^{k(j)} D_i^k s^{k(\ell)})$. The following proposition gives a necessary and sufficient condition to establish the existence of a dominance preserving order for a dominance relation.

Proposition 3 *Let D^k be a dominance relation on S^k. There exists a dominance preserving order for D^k if and only if D^k does not admit cycles in its asymmetric part.*

Proof: $\Rightarrow$ The existence of a cycle in the asymmetric part of D^k would imply the existence of two consecutive states $s^{k(j)}$ and $s^{k(\ell)}$ on this cycle with $j > \ell$, a contradiction.
$\Leftarrow$ Any topological order based on the asymmetric part of D^k is a dominance preserving order for D^k. □

We give in section 4.3.1 an example of a dominance preserving order. If states of C_{i-1}^k are generated according to a dominance preserving order for D_i^k, step 6 and loop 9-11 of Algorithm 2 can be omitted.

4 Implementation issues

We first present the order in which we consider items in the sequential process (section 4.1). Then, we present three dominance relations that we use in DP (section 4.2) and the way of applying them (section 4.3).

4.1 Item order

The order in which items are considered is a crucial implementation issue in DP. In the single-objective knapsack problem, it is well-known that, in order to obtain a good solution, items should usually be considered in decreasing order of value to weight ratios v^k/w^k (assuming that ties are solved arbitrarily) [3, 4]. For the multi-objective version, there is no such a natural order.

We introduce now three orders $\mathcal{O}^{sum}$, $\mathcal{O}^{\max}$, $\mathcal{O}^{\min}$ that are derived by aggregating orders $\mathcal{O}^i$ induced by the ratios v_i^k/w^k for each criterion ($i = 1, \ldots, p$). Let r_i^ℓ be the rank or position of item ℓ in order $\mathcal{O}^i$. $\mathcal{O}^{sum}$ denotes an order according to increasing values of the sum of the ranks of items in the p orders $\mathcal{O}^i$ ($i = 1, \ldots, p$). $\mathcal{O}^{\max}$ denotes an order according to the increasing values of the maximum or worst rank of items in the p orders $\mathcal{O}^i$ ($i = 1, \ldots, p$), where the worst rank of item ℓ in the p orders $\mathcal{O}^i$ ($i = 1, \ldots, p$) is computed by $\max_{i=1,\ldots,p}\{r_i^\ell\} + \frac{1}{pn}\sum_{i=1}^p r_i^\ell$ in order to discriminate items with the same maximum rank. $\mathcal{O}^{\min}$ denotes an order according to the increasing values of the minimum or best rank of items in the p orders $\mathcal{O}^i$ ($i = 1, \ldots, p$), where the best rank of item ℓ in the p orders $\mathcal{O}^i$ ($i = 1, \ldots, p$) is computed by $\min_{i=1,\ldots,p}\{r_i^\ell\} + \frac{1}{pn}\sum_{i=1}^p r_i^\ell$ in order to discriminate items with the same minimum rank.

In the computational experiments, in Section 5.2.1, we show the impact of the order on the efficiency of our approach.

4.2 Dominance relations

Each dominance relation focuses on specific considerations. It is then desirable to make use of complementary dominance relations. Moreover, when deciding to use a dominance relation, a tradeoff must be made between its potential ability of discarding many states and the time it requires to be checked.

We present now the three dominance relations used in our method. The first two relations are very easy to establish and the last one, although more difficult to establish, is considered owing to its complementarity with the two others.

We first present a dominance relation based on the following observation. When the residual capacity associated to a state s^k of phase k is greater than or equal to the sum of the weights of the remaining items (items $k+1, \ldots, n$), the only completion of s^k that can possibly lead to an efficient solution is the full completion $J = \{k+1, \ldots, n\}$. Thus, in this context, it is unnecessary to generate extensions of s^k that do not contain all the remaining items. We define thus the dominance relation D_r^k on S^k for $k = 1, \ldots, n$ by:

$$\text{for all } s^k, \tilde{s}^k \in S^k,\ s^k D_r^k \tilde{s}^k \Leftrightarrow \begin{cases} \tilde{s}^k \in S^{k-1}, \\ s^k = (\tilde{s}_1^k + v_1^k, \ldots, \tilde{s}_p^k + v_p^k, \tilde{s}_{p+1}^k + w^k), \text{ and} \\ \tilde{s}_{p+1}^k \leq W - \sum_{j=k}^n w^j \end{cases}$$

The following proposition shows that D_r^k is indeed a dominance relation and gives additional properties of D_r^k.

Proposition 4 (Relation D_r^k)

(a) D_r^k is a dominance relation

(b) D_r^k is transitive

(c) D_r^k admits dominance preserving orders

Proof: (a) Consider two states s^k and $\tilde{s}^k$ such that $s^k D_r^k \tilde{s}^k$. This implies, that $s^k \underline{\Delta} \tilde{s}^k$. Moreover, since $s_{p+1}^k = \tilde{s}_{p+1}^k + w^k \leq W - \sum_{j=k+1}^n w^j$, any subset $J \subseteq \{k+1, \ldots, n\}$ is a completion for $\tilde{s}^k$ and s^k. Thus, for all $\tilde{s}^n \in \text{Ext}(\tilde{s}^k)$, there exists $s^n \in \text{Ext}(s^k)$, based on the same completion as $\tilde{s}^n$, such that $s^n \underline{\Delta} \tilde{s}^n$. This establishes that D_r^k satisfies condition (1) of Definition 2.
(b) Obvious.
(c) By Proposition 3, since D_r^k is transitive. □

This dominance relation is rather poor, since at each phase k it can only appear between a state that does not contain item k and its extension that contains item k. Nevertheless, it is very easy to check since, once the residual capacity $W - \sum_{j=k}^n w^j$ is computed, relation D_r^k requires only one test to be established between two states.

We present now dominance relation $D_{\underline{\Delta}}^k$ that is a generalization to the multi-objective case of the dominance relation usually attributed to Weingartner and Ness [18] and used in the classical Nemhauser and Ullmann's algorithm [19]. This second dominance relation is defined on S^k for $k = 1, \ldots, n$ by:

$$\text{for all } s^k, \tilde{s}^k \in S^k,\ s^k D_{\underline{\Delta}}^k \tilde{s}^k \Leftrightarrow \begin{cases} s^k \underline{\Delta} \tilde{s}^k \text{ and} \\ s_{p+1}^k \leq \tilde{s}_{p+1}^k \text{ if } k < n \end{cases}$$

Observe that the condition on the weights s_{p+1}^k and $\tilde{s}_{p+1}^k$ ensures that every completion for $\tilde{s}^k$ is also a completion for s^k. The following proposition shows that $D_{\underline{\Delta}}^k$ is indeed a dominance relation and gives additional properties of $D_{\underline{\Delta}}^k$.

Proposition 5 (Relation $D_{\underline{\Delta}}^k$)

(a) $D_{\underline{\Delta}}^k$ is a dominance relation

(b) $D^k_{\underline{\Delta}}$ *is transitive*

(c) $D^k_{\underline{\Delta}}$ *admits dominance preserving orders*

(d) $D^n_{\underline{\Delta}} = \underline{\Delta}$

Proof: (a) Consider two states s^k and $\tilde{s}^k$ such that $s^k D^k_{\underline{\Delta}} \tilde{s}^k$. This implies, that $s^k \underline{\Delta} \tilde{s}^k$. Moreover, since $s^k_{p+1} \leq \tilde{s}^k_{p+1}$, any subset $J \subseteq \{k+1, \ldots, n\}$ that is a completion for $\tilde{s}^k$ is also a completion for s^k. Thus, for all $\tilde{s}^n \in \text{Ext}(\tilde{s}^n)$, there exists $s^n \in \text{Ext}(s^n)$, based on the same completion as $\tilde{s}^n$, such that $s^n \underline{\Delta} \tilde{s}^n$. This establishes that $D^k_{\underline{\Delta}}$ satisfies condition (1) of Definition 2.
(b) Obvious.
(c) By Proposition 3, since $D^k_{\underline{\Delta}}$ is transitive.
(d) By definition. □

Relation $D^k_{\underline{\Delta}}$ is a powerful relation since a state can possibly dominate all other states of larger weight. This relation requires at most $p+1$ tests to be established between two states.

The third dominance relation is based on the comparison between specific extensions of a state and an upper bound of the extensions of another state. An upper bound for a state is defined as follows in our context.

Definition 3 (Upper bound) *Criterion vector* $u = (u_1, \ldots, u_p)$ *is an upper bound for a state* $s^k \in S^k$ *if and only if for all* $s^n \in Ext(s^k)$ *we have* $u_i \geq s^n_i$, $i = 1, \ldots, p$.

We can derive a general type of dominance relations as follows: considering two states $s^k, \tilde{s}^k \in S^k$, if there exists a completion J of s^k and an upper bound $\tilde{u}$ for $\tilde{s}^k$ such that $s^k_i + \sum_{j \in J} v^j_i \geq \tilde{u}_i$, $i = 1, \ldots, p$, then s^k dominates $\tilde{s}^k$.

This type of dominance relations can be implemented only for specific completions and upper bounds. In our experiments, we just consider two specific completions J' and J'' obtained by a simple greedy algorithm as follows. After relabeling items $k+1, \ldots, n$ according to order $\mathcal{O}^{sum}$ (respectively, $\mathcal{O}^{\max}$), completion J' (respectively, J'') is obtained by inserting sequentially the remaining items into the solution provided that the capacity constraint is respected.

To compute u, we use the upper bound presented in [3] for each criterion value. Let us first define $\overline{W}(s^k) = W - s^k_{p+1}$ the residual capacity associated to state $s^k \in S^k$. We denote by $c_i = \min\{\ell_i \in \{k+1, \ldots, n\} : \sum_{j=k+1}^{\ell_i} w^j > \overline{W}(s^k)\}$ the position of the first item that cannot be added to state $s^k \in S^k$ when items $k+1, \ldots, n$ are relabeled according to order $\mathcal{O}^i$. Thus, according to [3, Th 2.2], when items $k+1, \ldots, n$ are relabeled according to order $\mathcal{O}^i$, an upper bound on the ith criterion value of $s^k \in S^k$ is for $i = 1, \dots, p$:

$$u_i = s_i^k + \sum_{j=k+1}^{c_i - 1} v_i^j + \max\left\{ \left\lfloor \overline{W}(s^k) \frac{v_i^{c_i+1}}{w^{c_i+1}} \right\rfloor, \left\lfloor v_i^{c_i} - (w^{c_i} - \overline{W}(s^k)) \frac{v_i^{c_i-1}}{w^{c_i-1}} \right\rfloor \right\} \tag{2}$$

Finally, we define D_b^k a particular dominance relation of this general type for $k = 1, \dots, n$ by:

$$\text{for all } s^k, \tilde{s}^k \in S^k,\ s^k D_b^k \tilde{s}^k \Leftrightarrow \begin{cases} s_i^k + \sum_{j \in J'} v_i^j \geq \tilde{u}_i, & i = 1, \dots, p \\ \text{or} \\ s_i^k + \sum_{j \in J''} v_i^j \geq \tilde{u}_i, & i = 1, \dots, p \end{cases}$$

where $\tilde{u} = (\tilde{u}_1, \dots, \tilde{u}_p)$ is the upper bound for $\tilde{s}^k$ computed according to (2).

The following proposition shows that D_b^k is indeed a dominance relation and gives additional properties of D_b^k.

Proposition 6 (Relation D_b^k)

(a) D_b^k is a dominance relation

(b) D_b^k is transitive

(c) D_b^k admits dominance preserving orders

(d) $D_b^n = \underline{\Delta}$

Proof: (a) Consider states s^k and $\tilde{s}^k$ such that $s^k D_b^k \tilde{s}^k$. This implies that there exists $J \in \{J', J''\}$ leading to an extension s^n of s^k such that $s^n \underline{\Delta} \tilde{u}$. Moreover, since $\tilde{u}$ is an upper bound of $\tilde{s}^k$, we have $\tilde{u} \underline{\Delta} \tilde{s}^n$, for all $\tilde{s}^n \in \text{Ext}(\tilde{s}^k)$. Thus, by transitivity of $\underline{\Delta}$, we get $s^n \underline{\Delta} \tilde{s}^n$, which establishes that D_b^k satisfies condition (1) of Definition 2.
(b) Consider states s^k, $\tilde{s}^k$, and $\bar{s}^k$ such that $s^k D_b^k \tilde{s}^k$ and $\tilde{s}^k D_b^k \bar{s}^k$. This implies that, on the one hand, there exists $J_1 \in \{J', J''\}$ such that $s_i^k + \sum_{j \in J_1} v_i^j \geq \tilde{u}_i$ $(i = 1, \dots, p)$, and on the other hand, there exists $J_2 \in \{J', J''\}$ such that $\tilde{s}_i^k + \sum_{j \in J_2} v_i^j \geq \bar{u}_i$ $(i = 1, \dots, p)$. Since $\tilde{u}$ is an upper bound for $\tilde{s}^k$ we have $\tilde{u}_i \geq \tilde{s}_i^k + \sum_{j \in J_2} v_i^j$ $(i = 1, \dots, p)$. Thus we get $s^k D_b^k \bar{s}^k$.
(c) By Proposition 3, since D_b^k is transitive.
(d) By definition. □

D_b^k is harder to check than relations D_r^k and $D_{\underline{\Delta}}^k$ since it requires much more tests and state-dependent information.

Obviously, relation D_b^k would have been richer if we had used additional completions (according to other orders) for s^k and computed instead of one upper bound u, an upper bound set using, *e.g.*, the techniques presented in [20]. Nevertheless, in our context since we have to check D_b^k for many states, enriching D_b^k in this way would be extremely time consuming.

4.3 Implementing with multiple dominance relations

In order to be efficient, we will use the three dominance relations presented in section 4.2 at each phase. As underlined in the previous subsection, dominance relations require more or less computational effort to be checked. Moreover, even if they are partly complementary, it often happens that several relations are valid for a same pair of states. It is thus natural to apply first dominance relations which can be checked easily (such as D_r^k and $D_{\underline{\Delta}}^k$) and then test on a reduced set of states dominance relations requiring a larger computation time (such as D_b^k).

We describe now the details of the implementation of these dominance relations. Algorithm 3, which computes, at each phase k, the subset of candidates C^k from subset C^{k-1} ($k = 1, \ldots, n$), replaces step 3 to step 4 of Algorithm 1.

The use of relation D_r^k and $D_{\underline{\Delta}}^k$ is first described (steps 1-8) and then the use of relation D_b^k (steps 9-24). This algorithm uses two subprocedures: procedure `MaintainNonDominated`, which removes states $D_{\underline{\Delta}}^k$-dominated, and procedure `KeepNonDominated`, which is used during the application of relation D_b^k.

4.3.1 Generation of C_0^k and dominating preserving order

Generating *a priori* C_0^k and, then, trimming it using dominance relations in order to produce C^k would be inefficient. Instead, we generate and trim C_0^k progressively, which requires generating new states of C_0^k according to a dominance preserving order for $D_{\underline{\Delta}}^k$.

Let relation $\underline{\Delta}_{lex}$ denote the lexicographic relation defined on S^k by: for all $s^k, \tilde{s}^k \in S^k$, $s^k \underline{\Delta}_{lex} \tilde{s}^k \Leftrightarrow s_j^k > \tilde{s}_j^k$ where $j = \min\{i \in \{1, \ldots, p\} : s_i^k \neq \tilde{s}_i^k\}$ or $s_j^k = \tilde{s}_j^k$, $j = 1, \ldots, p$. Its asymmetric part is denoted by Δ_{lex}. Let relation $\geq_{lex}$ denote the lexicographic relation defined on S^k by: for all $s^k, \tilde{s}^k \in S^k$, $s^k \geq_{lex} \tilde{s}^k \Leftrightarrow s_{p+1}^k < \tilde{s}_{p+1}^k$ or ($s_{p+1}^k = \tilde{s}_{p+1}^k$ and $s^k \underline{\Delta}_{lex} \tilde{s}^k$). Its asymmetric part is denoted by $>_{lex}$.

Proposition 7 *The decreasing order with respect to $\geq_{lex}$ is a dominance preserving order for $D_{\underline{\Delta}}^k$.*

Proof: Consider a set $H = \{s^{k(1)}, \ldots, s^{k(h)}\} \subseteq S^k$ ordered according to decreasing order with respect to $\geq_{lex}$, *i.e.* such that $s^{k(i)} \geq_{lex} s^{k(j)}$ for all $i < j$ ($1 \leq i, j \leq h$). Suppose that H is not ordered according to a dominance preserving order for $D_{\underline{\Delta}}^k$. There exists thus $s^{k(i)}, s^{k(j)} \in H$ ($i < j, 1 \leq i, j \leq h$) such that $s^{k(j)} D_{\underline{\Delta}}^k s^{k(i)}$ and not($s^{k(i)} D_{\underline{\Delta}}^k s^{k(j)}$). Then, we have either $s_{p+1}^{k(j)} < s_{p+1}^{k(i)}$ or ($s_{p+1}^{k(j)} = s_{p+1}^{k(i)}$ and $s^{k(j)} \Delta_{lex} s^{k(i)}$). This implies that $s^{k(j)} >_{lex} s^{k(i)}$, which contradicts that H is ordered according to decreasing order with respect to $\geq_{lex}$. □

Observe also that $\geq_{lex}$ is trivially a dominance preserving order for D_r^k.

Our implementation maintains set C^k, $k = 1, \ldots, n$, sorted according to decreasing order with respect to $\geq_{lex}$. Considering indeed that, at phase k, C^{k-1} is sorted according to de-

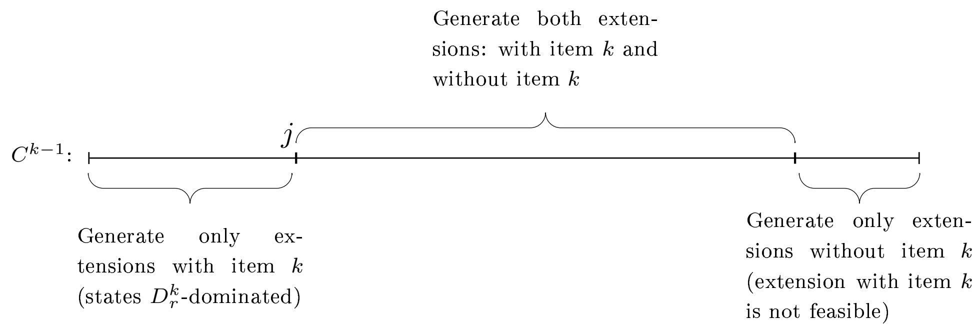
Generate both extensions: with item k and without item k
C^{k-1}:
j
Generate only extensions with item k (states D_r^k-dominated)
Generate only extensions without item k (extension with item k is not feasible)

Figure 1: Extensions of C^{k-1} (sorted according to $\geq_{lex}$)

creasing order with respect to $\geq_{lex}$, we generate progressively states of C_0^k according to this dominance preserving order, and thus maintain C^k sorted according to the same order.

4.3.2 Application of the three relations

We present now a detailed description of the application of each dominance relation D_r^k, $D_{\underline{\Delta}}^k$, and D_b^k. Generating C_1^k from C_0^k using D_r^k, then reducing C_1^k to C_2^k using $D_{\underline{\Delta}}^k$, and finally reducing C_2^k to C_3^k using D_b^k would not be computationally efficient. Instead, since D_r^k-dominated states can be identified in C^{k-1}, we generate directly C_2^k using $D_{\underline{\Delta}}^k$ and reduce C_2^k to C_3^k using D_b^k. In the following, we shall not distinguish sets C_i^k, $i = 0, \ldots, 3$, but instead refer to a current set C^k which is progressively reduced.

Application of relation D_r^k: The order of states in C^{k-1} allows us to find easily j, the index of the first state that is not D_r^k-dominated (step 2). Thus, it is unnecessary to generate the extension without item k for all states $s^{k-1(1)}, \ldots, s^{k-1(j-1)}$ since they are D_r^k-dominated by their respective extensions with item k. Figure 1 shows the extensions generated for each state of C^{k-1}, due to D_r^k on the one hand, and to infeasibility on the other hand.

Application of relation $D_{\underline{\Delta}}^k$: Since states are generated progressively according to a dominance preserving order for $D_{\underline{\Delta}}^k$, we never remove states from C^k. Indeed, by definition of a dominance preserving order, a state candidate to be added in C^k cannot $D_{\underline{\Delta}}^k$-dominate states already included in C^k.

In order to test efficiently $D_{\underline{\Delta}}^k$-dominance within C^k, we maintain $M^k \subseteq C^k$, the subset of non-dominated states of C^k with respect to profit values only. Indeed, since states are generated according to a dominance preserving order for $D_{\underline{\Delta}}^k$, a new state s^k can be $D_{\underline{\Delta}}^k$-dominated in C^k if and only if there exists a state in the current set M^k that $\underline{\Delta}$-dominates s^k considering that all states already generated have smaller or equal weight than s^k. This property is useful since M^k is much smaller than C^k (in the worst case, cardinality of C^k is in $O(|M^k| \times W)$). We impose that set M^k is sorted according to decreasing order with respect to

Algorithm 3: Computing C^k from C^{k-1} at each phase k $(k = 1, \dots, n)$

```
In  : $C^{k-1} = \{s^{k-1(1)}, \dots, s^{k-1(r)}\}$ such that $s^{k-1(i)} \geq_{lex} s^{k-1(j)}$ for all $i < j$ $(1 \leq i, j \leq r)$
Out: $C^k$ in which states are sorted according to decreasing preference with respect to $\geq_{lex}$
/* Application of relations $D^k_r$ and $D^k_{\underline{\Delta}}$ */
1  $C^k \leftarrow \emptyset$ ; $M^k \leftarrow \emptyset$ ; $i \leftarrow 1$ ; $j \leftarrow 1$ ;
/* Identification of $j$, index of the first state that is not $D^k_r$-dominated */
2  while $j \leq r$ and $s^{k-1(j)}_{p+1} + \sum_{\ell=k}^{n} w^\ell \leq W$ do $j \leftarrow j + 1$ ;
3  while $i \leq r$ and $s^{k-1(i)}_{p+1} + w^k \leq W$ do
4      $s^k \leftarrow (s^{k-1(i)}_1 + v^k_1, \dots, s^{k-1(i)}_p + v^k_p, s^{k-1(i)}_{p+1} + w^k)$;
5      while $j \leq r$ and $s^{k-1(j)} \geq_{lex} s^k$ do
6          MaintainNonDominated($s^{k-1(j)}$,$M^k$,$C^k$) ; $j \leftarrow j + 1$ ;
7      MaintainNonDominated($s^k$,$M^k$,$C^k$) ; $i \leftarrow i + 1$;
8  while $j \leq r$ do MaintainNonDominated($s^{k-1(j)}$,$M^k$,$C^k$) ; $j \leftarrow j + 1$;
/* Application of relation $D^k_b$ on $M^k \times C^k$ */
9  if $k = n$ then $C^n \leftarrow M^n$
10 else
11     $F \leftarrow \emptyset$;
       /* Generation of extensions $J'$ and $J''$ for each state of $M^k$ */
12     for order $\mathcal{O}$ in $\{\mathcal{O}^{sum}, \mathcal{O}^{\max}\}$ do
13         foreach $s^k \in M^k$ do
14             Relabel items $k+1, \dots, n$ according to order $\mathcal{O}$ ; $s^n \leftarrow s^k$;
15             for $j \leftarrow k + 1$ to $n$ do
16                 if $s^n_{p+1} + w^j \leq W$ then $s^n \leftarrow (s^n_1 + v^j_1, \dots, s^n_p + v^j_p, s^n_{p+1} + w^j)$;
17             $F \leftarrow$ KeepNonDominated($s^n$,$F$);
       /* Assuming that $C^k = \{s^{k(1)}, \dots, s^{k(c)}\}$ and that $F^n = \{s^{n(1)}, \dots, s^{n(h)}\}$ such that
          $s^{n(i)} \underline{\Delta}_{lex} s^{n(j)}$ for all $i < j$ $(1 \leq i, j \leq h)$ */
18     $i \leftarrow 1$ ; remove$\leftarrow$ true;
19     while $i \leq c$ and remove do
20         Compute an upper bound $u$ for $s^{k(i)}$ according to (2);
21         $j \leftarrow 1$ ; remove$\leftarrow$ false;
22         while $j \leq h$ and $s^{n(j)} \underline{\Delta}_{lex} u$ and not(remove) do
23             if $s^{n(j)} \underline{\Delta} u$ then remove $\leftarrow$ true else $j \leftarrow j + 1$;
24         if remove then $C^k \leftarrow C^k \backslash \{s^{k(i)}\}$ ; $i \leftarrow i + 1$ ;
25 return $C^k$
```

$\underline{\Delta}_{lex}$. The order of M^k is maintained easily by updating the sorted structure at each insertion.

```
Procedure MaintainNonDominated($s^k$,$M^k$,$C^k$)
/* Assume that $M^k = \{\tilde{s}^{k(1)}, \dots, \tilde{s}^{k(\ell)}\}$ such that $\tilde{s}^{k(i)} \underline{\Delta}_{lex} \tilde{s}^{k(j)}$ for all $i < j$ $(1 \leq i, j \leq \ell)$ */
1 $i \leftarrow 1$ ; dominated $\leftarrow$ false;
2 while $i \leq \ell$ and $\tilde{s}^{k(i)} \underline{\Delta}_{lex} s^k$ and not(dominated) do
3     if $\tilde{s}^{k(i)} \underline{\Delta} s^k$ then dominated $\leftarrow$ true else $i \leftarrow i + 1$;
4 if not(dominated) then
5     $C^k \leftarrow C^k \cup \{s^k\}$ ; // Insertion at the end of $C^k$
6     $M^k \leftarrow M^k \cup \{s^k\}$; // Insertion at the $i$th position in $M^k$
7     while $i \leq \ell$ do
8         if $s^k \underline{\Delta} \tilde{s}^{k(i)}$ then $M^k \leftarrow M^k \backslash \{\tilde{s}^{k(i)}\}$;
9         $i \leftarrow i + 1$;
```

In the bi-objective case, based on the idea of [21], using an AVL tree for storing states of M^k also leads to a significant improvement of the running time. The AVL tree allows us to perform each search, insertion or deletion in $O(\log |M^k|)$. With this structure the while loop 2-3 of procedure `MaintainNonDominated` reduces to the search of the largest value $i^\star \in \{1, \dots, \ell\}$ such that $\tilde{s}_1^{k(i^\star)} \geq s_1^k$. Then variable *dominated* is false if and only if $\tilde{s}_2^{k(i^\star)} < s_2^k$. Moreover, the while loop 7-9 of procedure `MaintainNonDominated` reduces to removing, from M^k, state $\tilde{s}^{k(i^\star)}$ if $\tilde{s}_1^{k(i^\star)} = s_1^k$ and states with index $i^\star + 1$ to $j^\star - 1$ where $j^\star \in \{i^\star, \dots, \ell\}$ is the smallest value such that $\tilde{s}_2^{k(j^\star)} > s_2^k$.

Thus, in the bi-objective case, the running time of procedure `MaintainNonDominated` can be bounded by $O(z \times \log |C_0^k|)$ where z represents the number of states that have to be removed from M^k. For $p > 2$, a linked list has to be used for storing M^k and the running time can be bounded by $O(|C_0^k|)$ only. Since in the worst case at most $|C_0^k|$ states have to be inserted in M^k and at most $|C_0^k| - 1$ states have to be deleted from M^k in the entire execution of Algorithm 3, the execution time of all calls of procedure `MaintainNonDominated`, during phase k, is in $O(|C_0^k| \log |C_0^k|)$ for $p = 2$ and in $O(|C_0^k|^2)$ for $p > 2$.

Application of relation D_b^k: Relation D_b^k is applied after relations D_r^k and $D_{\underline{\Delta}}^k$ to reduce the set C^k. The purpose of using this relation is to remove states of C^k with small weight since D_r^k and $D_{\underline{\Delta}}^k$ are not efficient to remove these states (for instance using $D_{\underline{\Delta}}^k$ we will never remove the empty knapsack). Thus, we test if states of small weight are D_b^k-dominated. We apply relation D_b^k between states of M^k, which contains states with non-dominated criterion vectors, and the current C^k. A state $s^{k(i)}$ of C^k is removed if there exists a state $s^k \neq s^{k(i)}$ in M^k such that $s^k D_b^k s^{k(i)}$. To do that, we generate two extensions for all states of M^k with respect to orders $\mathcal{O}^{sum}$ and $\mathcal{O}^{max}$ and keep only the non-dominated extensions in F. This is done by procedure `KeepNonDominated` that is not detailed here since it is just a simplified version of procedure `MaintainNonDominated` where F replaces M^k and step 5 is removed.

Then $s^{k(i)}$ is D_b^k-dominated by a state of M^k, if there exists $s^n \in F$ such that $s^n \underline{\Delta} u$, where u is the upper bound associated to $s^{k(i)}$. Since computing the upper bound for each state is time consuming, we stop checking relation D_b^k as soon as we identify a state of C^k that is not D_b^k-dominated by a state of M^k.

Special case of phase n First observe that, since $D_{\underline{\Delta}}^n = D_b^n = \underline{\Delta}$, it is unnecessary to apply both relations. Thus, due to the order of application of these relations ($D_{\underline{\Delta}}^k$ followed by D_b^k), we do not apply relation D_b^k at phase n.

Second, at phase n it should be noticed that M^n corresponds to the non-dominated criterion vectors of S^n and thus we take C^n equal to M^n (step 9).

5 Computational experiments and results

5.1 Experimental design

All experiments presented here were performed on a bi-Xeon 3.4GHz with 3072Mb RAM. All algorithms are written in C++. In the bi-objective case ($p = 2$), the following types of instances were considered:

A) Random instances: $v_1^k \in_R [1, 1000]$, $v_2^k \in_R [1, 1000]$ and $w^k \in_R [1, 1000]$

B) Unconflicting instances, where v_1^k is positively correlated with v_2^k: $v_1^k \in_R [111, 1000]$ and $v_2^k \in_R [v_1^k - 100, v_1^k + 100]$, and $w^k \in_R [1, 1000]$

C) Conflicting instances, where v_1^k and v_2^k are negatively correlated: $v_1^k \in_R [1, 1000]$, $v_2^k \in_R [\max\{900 - v_1^k; 1\}, \min\{1100 - v_1^k; 1000\}]$, and $w^k \in_R [1, 1000]$

D) Conflicting instances with correlated weight, where v_1^k and v_2^k are negatively correlated, and w^k is positively correlated with v_1^k and v_2^k: $v_1^k \in_R [1, 1000]$, $v_2^k \in_R [\max\{900 - v_1^k; 1\}, \min\{1100 - v_1^k; 1000\}]$, and $w^k \in_R [v_1^k + v_2^k - 200; v_1^k + v_2^k + 200]$.

where $\in_R [a, b]$ denotes uniformly random generated in $[a, b]$. For all these instances, we set $W = \lfloor 1/2 \sum_{k=1}^{n} w^k \rfloor$.

Most of the time in the literature, experiments are only made on instances of type A. Sometimes, other instances such as those of type B, which were introduced in [11], are studied. However, instances of type B should be viewed as quasi single-criterion instances since they involve two non conflicting criteria. This aspect can be seen in Figure 2. Nevertheless, in a bi-objective context, considering conflicting criteria is a more appropriate way of modeling real-world situations. For this reason, we introduced instances of types C and D for which criterion values of items are conflicting. In this case, items are located around the line $y = -x + 1000$. In instances of type D, w^k is positively correlated with v_1^k, v_2^k. These instances were introduced in order to verify if positively correlated instances are harder than uncorrelated instances as in the single-criterion context [4].

For three-objective experiments, we considered the generalization of random instances of type A where $v_i^k \in_R [1, 1000]$ for $i = 1, \ldots, 3$ and $w^k \in_R [1, 1000]$ and the generalization of conflicting instances of type C where $v_1^k \in_R [1, 1000]$, $v_2^k \in_R [1, 1001 - v_1^k]$, and $v_3^k \in_R [\max\{900 - v_1^k - v_2^k; 1\}, \min\{1100 - v_1^k - v_2^k; 1001 - v_1^k\}]$, and $w^k \in_R [1, 1000]$.

For each type of instances and each value of n presented in this study, 10 different instances were generated. In the following, we denote by pTn a p criteria instance of type T with n items. For example 2A100 denotes a bi-objective instance of type A with 100 items.

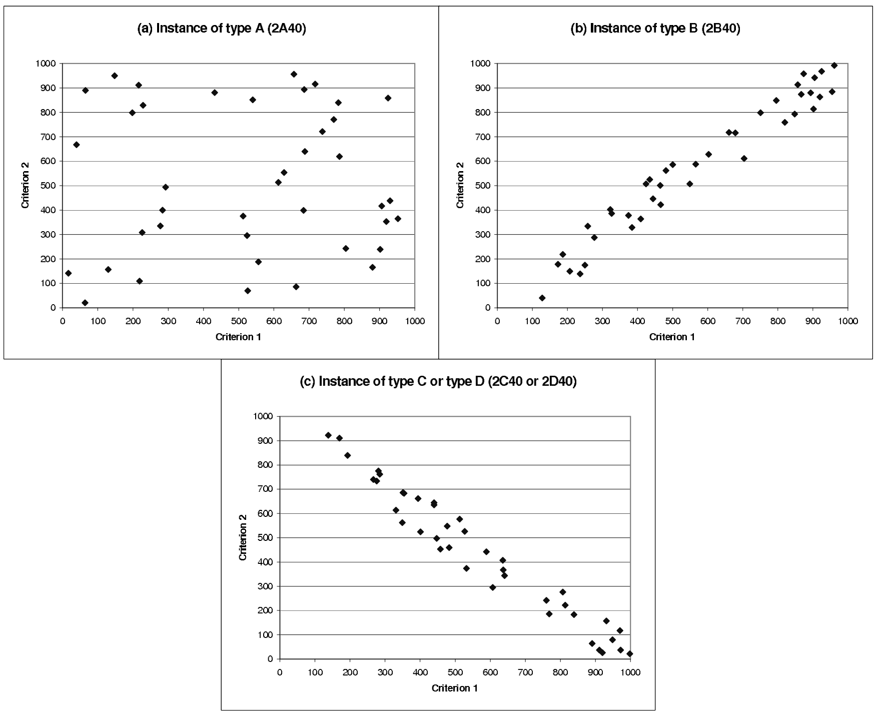

Figure 2: Repartition in the criterion space of values of items for one instance of each type

5.2 Results in the bi-objective case

The goals of the experiments in the bi-objective case are:

(a) to determine the best order to sort items in our approach (section 5.2.1)

(b) to evaluate the cardinality of the set of non-dominated criterion vectors on different types of instances (section 5.2.2)

(c) to analyze the impact of using dominance relations $D_{\underline{\Delta}}^k$, D_b^k, and D_r^k (section 5.2.3)

(d) to analyze the performance of our approach on large size instances (section 5.2.4)

(e) to compare our approach with other exact methods (section 5.2.5)

5.2.1 Item order

Table 1: Impact of different orders of items in our approach (Average CPU time in seconds)

Type	n	$\mathcal{O}^{\max}$		$\mathcal{O}^{sum}$		$\mathcal{O}^{\min}$		Random
A	300	84.001	(-53%)	100.280	(-44%)	94.598	(-47%)	178.722
B	600	1.141	(-99%)	1.084	(-99%)	1.403	(-98%)	77.699
C	200	59.986	(-44%)	60.061	(-44%)	85.851	(-20%)	107.973
D	90	20.795	(-34%)	23.687	(-25%)	35.426	$(+12\%)$	31.659

The increase or the decrease (expressed in percent) of CPU time compared to the CPU time obtained when items are selected randomly is given in brackets.

The way of ordering items has a dramatic impact on the CPU time, has shown in Table 1. We compare, on 10 instances of each type, the results obtained using the three orders presented in section 4.1 ($\mathcal{O}^{\max}$, $\mathcal{O}^{sum}$, and $\mathcal{O}^{\min}$) and results obtained with a random order of objects. Table 1 shows clearly that order $\mathcal{O}^{\max}$ is significantly better for all types of instances. Thus, in the following, items are sorted and labeled according to $\mathcal{O}^{\max}$.

5.2.2 Cardinality of the set of non-dominated criterion vectors

Figure 3 shows the evolution of the average cardinality of the set of non-dominated criterion vectors for 10 instances of each type. As expected, instances of type B are quasi single-objective instances and have very few non-dominated criterion vectors. Even if instances of type A have more non-dominated criterion vectors than instances of type B, the conflicting instances (type C and D) have many more non-dominated criterion vectors than the other types of instances.

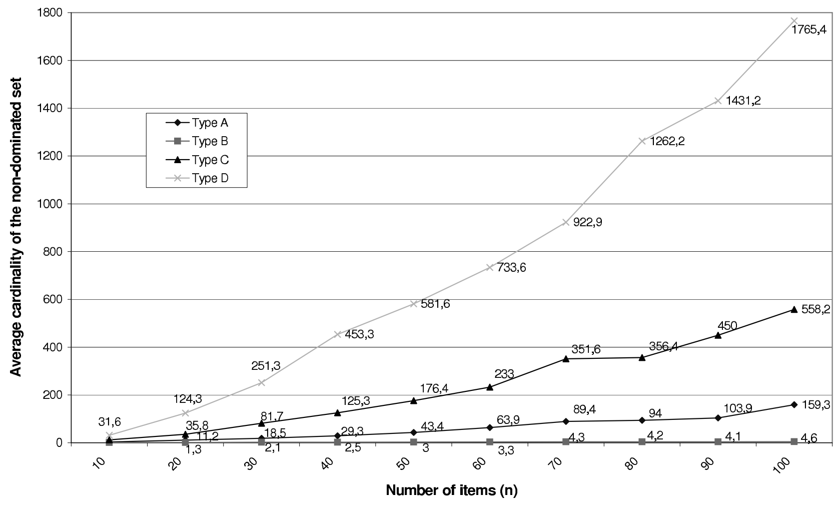

Figure 3: Average cardinality of the set of non-dominated criterion vectors as a function of n

5.2.3 Impact of each dominance relation

We compare, in Table 2, the average CPU time obtained using dominance relation $D^k_{\underline{\Delta}}$ alone, relations D^k_r and $D^k_{\underline{\Delta}}$, relations $D^k_{\underline{\Delta}}$ and D^k_b, and finally relations D^k_r, $D^k_{\underline{\Delta}}$, and D^k_b all together. Table 2 shows clearly that it is always better to use these three relations together, due to their complementarity. Thus, in the following experiments, we always apply these three relations together.

Table 2: Complementarity of dominance relations D^k_r, $D^k_{\underline{\Delta}}$, and D^k_b in our approach (Average CPU time in seconds)

Type	n	$D^k_{\underline{\Delta}}$	D^k_r and $D^k_{\underline{\Delta}}$		$D^k_{\underline{\Delta}}$ and D^k_b		D^k_r, $D^k_{\underline{\Delta}}$, and D^k_b	
A	300	272.628	157.139	(−42.4%)	85.076	(−68.8%)	84.001	(−69.2%)
B	600	230.908	174.015	(−24.6%)	1.188	(−99.5%)	1.141	(−99.5%)
C	200	122.706	63.557	(−48.2%)	61.696	(−49.7%)	59.986	(−51.1%)
D	90	46.137	24.314	(−47.3%)	23.820	(−48.4%)	20.795	(−54.9%)

The decrease (expressed in percent) of CPU time compared to the CPU time obtained when using only relation $D^k_{\underline{\Delta}}$ in our approach is given in brackets.

To illustrate further the impact and complementarity of each dominance relation, we indicate, in Table 3, the number of states respectively removed by relations D^k_r, $D^k_{\underline{\Delta}}$, and D^k_b for one instance (with $p = 2$, and $n = 20$) of each type. In addition, the number of non-feasible states obtained at each phase and the cardinality of C^k is given. For instance 2B20, the most efficient relation in terms of removed states is relation D^k_b. This is not surprising, since the values of the non-dominated extensions of a state s^k are not spread, and thus the upper bound for s^k, which is an upper bound on the ideal point associated to the extensions of s^k, is very close to the values of the extensions of s^k. However, even if this relation removes many states in all others instances, the most efficient relation, for the others instances, is relation $D^k_{\underline{\Delta}}$. It removes up to 4609 states in instance 2D20, whereas relations D^k_r and D^k_b remove respectively 504 and 1990 states, that is less states than the feasibility condition. For all instances, relation D^k_r is the least efficient. Nevertheless, this relation is extremely unexpensive in terms of CPU time. For instances of type C and D, even if relations $D^k_{\underline{\Delta}}$ and D^k_b remove the majority of the states, relation D^k_r removes a non negligible number of the states.

5.2.4 Results on large size instances

We present, in Table 4, results of our approach on large size instances of each type. The largest instances solved here are those of type B with 4000 items and the instances with the largest number of non-dominated criterion vectors are those of type D with 250 items for which the cardinality of the set of non-dominated criterion vectors is in average of 8154.7.

Table 3: Impact of D_r^k, $D_{\underline{\Delta}}^k$, and D_b^k on one instance of type A, B, C and D for $n = 20$

One instance 2A20 where $|ND| = 21$

Phase	# States				
	Removed by			non	in
	D_r^k	$D_{\underline{\Delta}}^k$	D_b^k	feasible	C^k
1	0	0	0	0	2
2	0	0	0	0	4
3	0	0	2	0	6
4	0	0	1	0	11
5	0	3	5	0	14
6	0	10	1	0	17
7	0	6	3	0	25
8	0	16	3	0	31
9	0	22	3	0	37
10	0	22	0	0	52
11	0	36	0	0	68
12	0	37	0	0	99
13	0	75	0	4	119
14	0	29	25	12	172
15	0	128	7	36	173
16	5	67	63	33	178
17	0	88	32	51	185
18	0	80	44	82	164
19	5	38	112	108	65
20	14	3	-	51	21
Total	24	660	301	377	-

One instance 2B20 where $|ND| = 1$

Phase	# States				
	Removed by			non	in
	D_r^k	$D_{\underline{\Delta}}^k$	D_b^k	feasible	C^k
1	0	0	1	0	1
2	0	0	1	0	1
3	0	0	1	0	1
4	0	0	1	0	1
5	0	0	1	0	1
6	0	0	1	0	1
7	0	0	0	0	2
8	0	0	2	0	2
9	0	0	2	0	2
10	0	0	1	0	3
11	0	0	0	0	6
12	0	3	0	2	7
13	0	0	2	2	10
14	0	0	7	6	7
15	0	2	2	3	7
16	0	0	4	7	3
17	0	0	0	3	3
18	0	0	0	3	3
19	0	0	2	3	1
20	0	0	-	1	1
Total	0	5	28	30	-

One instance 2C20 where $|ND| = 31$

Phase	# States				
	Removed by			non	in
	D_r^k	$D_{\underline{\Delta}}^k$	D_b^k	feasible	C^k
1	0	0	0	0	2
2	0	0	0	0	4
3	0	1	0	0	7
4	0	3	1	0	11
5	0	4	1	0	16
6	0	8	3	0	22
7	0	16	0	0	27
8	0	19	2	0	34
9	0	16	3	0	51
10	0	32	1	0	66
11	0	36	0	0	93
12	0	67	3	0	116
13	0	77	0	10	144
14	5	85	39	23	170
15	29	83	20	18	189
16	21	88	19	54	189
17	30	143	30	32	166
18	31	46	66	71	145
19	47	64	111	35	108
20	50	36	-	58	31
Total	213	824	299	301	-

One instance 2D20 where $|ND| = 189$

Phase	# States				
	Removed by			non	in
	D_r^k	$D_{\underline{\Delta}}^k$	D_b^k	feasible	C^k
1	0	0	0	0	2
2	0	0	0	0	4
3	0	0	0	0	8
4	0	0	0	0	16
5	0	10	0	0	22
6	0	3	0	0	41
7	0	9	0	0	73
8	0	51	0	0	95
9	0	24	0	0	166
10	0	46	0	1	285
11	1	66	1	7	495
12	8	243	0	13	726
13	29	356	67	59	941
14	16	415	60	84	1307
15	60	575	241	180	1558
16	19	1157	198	269	1473
17	131	699	412	379	1325
18	59	599	425	508	1059
19	113	308	586	446	665
20	68	48	-	597	189
Total	504	4609	1990	2543	-

We can observe that the results of Figure 3 concerning the size of the set of non-dominated criterion vectors are confirmed on large instances. The average maximum cardinality of C^k, which is a good indicator of the memory storage needed to solve the instances, can be very huge. This explains why we can only solve instances of type D up to 250 items.

Table 4: Results of our approach on large size instances

Type	n	Time in s.			$\|ND\|$			Avg
		Min	Avg	Max	Min	Avg	Max	$\max_k\{\|C^k\|\}$
A	100	0.152	0.328	0.600	98	159.3	251	17134.7
	200	6.768	12.065	21.025	416	529.0	729	209198.9
	300	57.475	84.001	101.354	905	1130.7	1651	898524.7
	400	243.215	307.093	369.999	1308	1713.3	2101	2230069.4
	500	677.398	889.347	1198.190	2034	2537.5	2997	5120514.7
	600	1833.080	2253.421	3116.670	2792	3593.9	4746	9983975.8
	700	4046.450	5447.921	7250.530	3768	4814.8	5939	18959181.7
B	1000	4.328	8.812	15.100	105	157.0	218	134107.2
	2000	139.836	251.056	394.104	333	477.7	630	1595436.1
	3000	1192.190	1624.517	2180.860	800	966.9	1140	6578947.2
	4000	4172.530	6773.264	8328.280	1304	1542.3	1752	18642759.0
C	100	1.564	2.869	4.636	406	558.2	737	103921.5
	200	43.834	59.986	93.541	1357	1612.8	2018	918162.6
	300	311.995	373.097	470.429	2510	2893.6	3297	3481238.4
	400	1069.290	1390.786	1670.500	3763	4631.8	5087	9400565.3
	500	2433.320	4547.978	6481.970	5111	7112.1	9029	21282280.5
D	100	36.450	40.866	54.267	1591	1765.4	2030	1129490.3
	150	235.634	265.058	338.121	2985	3418.5	3892	4274973.9
	200	974.528	1145.922	1497.700	4862	5464.0	6639	12450615.5
	250	2798.040	3383.545	3871.240	7245	8154.7	8742	26999714.8

5.2.5 Comparison with other exact methods

The results of a comparative study, in the bi-objective case, between the exact method of Captivo *et al.* [11], an exact method based on a commercial Integer Programming (IP) solver, and our approach using D^k_r, $D^k_{\underline{\Delta}}$, and D^k_b are presented in Table 5.

The Labeling Approach (LA) of Captivo *et al.* [11] was selected since it is the most efficient method currently known. An exact method, of the ε-constraint type [22], using a commercial IP solver was also considered for two major reasons. First, it is relatively easy to implement. Second, it has much less storage problems than the two other methods, since each efficient solution is found by solving one new 0-1 linear program. This ε-constraint method basically consists of optimizing the first criterion while moving iteratively a constraint on the second criterion. More precisely, in order to eliminate weakly efficient solutions, a slightly perturbed objective function is used relying on the fact that the criterion vectors are integer valued (see Algorithm 5). Cplex 9.0 is used as an IP solver in Algorithm 5 which is written in C++.

```
Algorithm 5: $\varepsilon$-constraint
1 Generate $y$ one optimal solution of $\max_{x\in X} f_1(x)$ ; Generate $z$ one optimal solution of $\max_{x\in X} f_2(x)$;
2 Generate $x^1$ one optimal solution of $\max\{f_2(x) : x \in X, f_1(x) \geq f_1(y)\}$;
3 $X^\star \leftarrow X^\star \cup \{x^1\}$ ; $j \leftarrow 1$;
4 while $f_2(x^j) < f_2(z)$ do
      /* optimize the function associated to the line passing through $(f_1(x^j), f_2(x^j))$ and
         $(f_1(x^j) - 1, f_2(z))$ subject to a restriction on the second objective */
5     Generate $x^{j+1}$ one optimal solution of
      $\max\{(f_2(z) - f_2(x^j))f_1(x) + f_2(x) : x \in X, f_2(x) \geq f_2(x^j) + 1\}$;
6     $X^\star \leftarrow X^\star \cup \{x^{j+1}\}$ ; $j \leftarrow j + 1$;
7 return $X^\star$;
```

The three methods have been used on the same instances and the same computer. For LA, we used the source code, in C, obtained from the authors. Table 5 presents results, in the bi-objective case, for instances of type A, B, C, and D for increasing size of n while LA can solve all instances of the series considered. Due to storage requirements, LA can only solve instances of type A up to 300 items, of type B up to 800 items, of type C up to 200 items, and of type D up to 100 items. As a comparison, we recall (see Table 4) that our approach can solve much larger size instances, respectively up to 700, 4000, 500, and 250 items.

Table 5: Comparison between the Labeling Approach (LA) of Captivo *et al.* [11], ε-constraint method and our approach.

Type	n	Avg time in s.					Avg
		LA	ε-constraint		Our approach		$\|ND\|$
A	100	2.476	5.343	(+116%)	0.328	(−87%)	159.3
	200	37.745	57.722	(+53%)	12.065	(−68%)	529.0
	300	163.787	285.406	(+74%)	84.001	(−49%)	1130.7
B	600	27.694	27.543	(−1%)	1.141	(−96%)	74.3
	700	47.527	29.701	(−38%)	2.299	(−95%)	78.6
	800	75.384	68.453	(−9%)	5.280	(−93%)	118.1
C	100	12.763	208.936	(+1537%)	2.869	(−78%)	558.2
	200	114.171	6584.012	(+5667%)	59.986	(−47%)	1612.8
D	100	127.911	23126.926	(+17980%)	40.866	(−68%)	1765.4

The decrease or increase (expressed in percent) of CPU time compared to the CPU time obtained with the Labeling Approach (LA) of Captivo *et al.* [11] is given in brackets.

Considering CPU time, we can conclude that our approach is always faster than LA and ε-constraint on the considered instances. Moreover, when the number of non-dominated criterion vectors increases, CPU time becomes prohibitive for ε-constraint (about 6.5 hours in average for instances 2D100), while storage limitations become restrictive for LA.

5.3 Results in the three-objective case

The goals of the experiments in the three-objective case are:

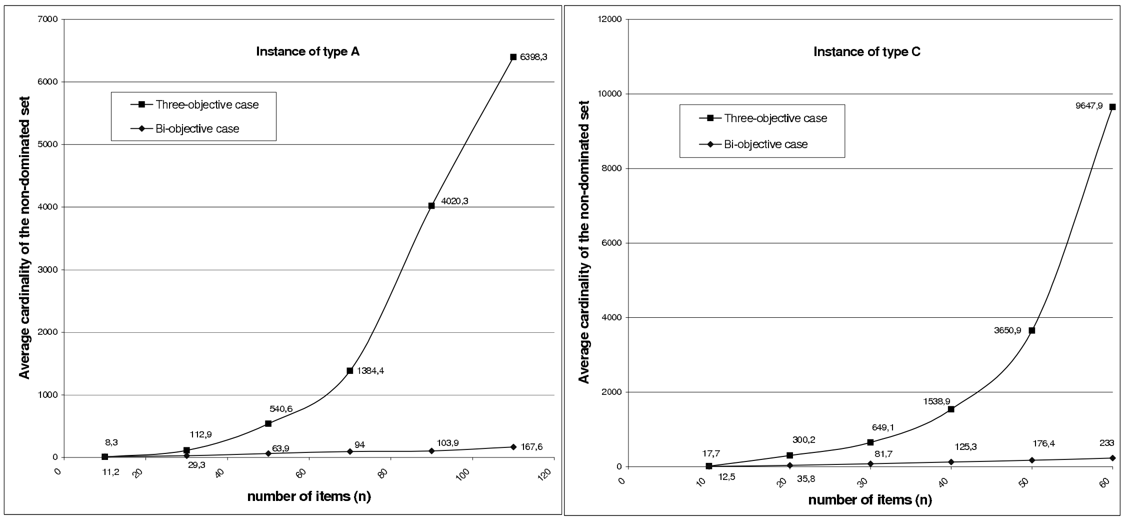

Figure 4: Evolution of the average cardinality of the set of non-dominated criterion vectors for instances of type A and C in the bi-objective and three-objective cases in function of n

(a) to evaluate the size of the set of non-dominated criterion vectors (see Figure 4 and Table 6)

(b) to analyze the performance of our approach on large instances (see Table 6)

We compare, in Figure 4, the evolution of the cardinality of the set of non-dominated criterion vectors in the bi-objective case and in the three-objective case for instances of type A and C. We can observe that the addition of one criterion leads to an explosion of the average cardinality of the set of non-dominated criterion vectors for both types of instances. For example, for $n = 50$ the increase is about a factor 8.5 for instances of type A and about 20.5 for instances of type C.

Table 6: Results of our approach on instances of types A and C in the three-objective case.

type	n	Time in s.			$\|ND\|$			Avg
		Min	Avg	Max	Min	Avg	Max	$\max_k\{\|C^k\|\}$
A	10	<1ms	<1ms	<1ms	4	8.3	18	20.9
	30	<1ms	0.012	0.028	31	112.9	193	1213.2
	50	0.112	0.611	1.436	266	540.6	930	12146.5
	70	4.204	16.837	44.858	810	1384.4	2145	64535.4
	90	80.469	538.768	2236.230	2503	4020.3	6770	285252.1
	110	273.597	3326.587	11572.700	3265	6398.3	9394	601784.6
C	10	<1ms	<1ms	0.004	5	17.7	32	53.4
	20	0.004	0.030	0.184	80	300.2	1270	1557.8
	30	0.016	0.431	2.076	72	649.1	2064	6861.1
	40	1.008	3.684	12.336	1167	1538.9	2740	23837
	50	4.840	83.594	316.811	1282	3650.9	6566	92155.4
	60	73.704	2572.981	13607.100	3698	9647.9	22713	328238.8

We present, in table 6, results of our approach concerning large size instances of types A and C in the three-objective case. Observe that the number of non-dominated criterion vectors varies a lot. This explains the variation of the CPU time which is strongly related with the number of non-dominated criterion vectors. Table 6 confirms for the three-objective case that instances of type A are easier to solve than instances of type C, as in the bi-objective case.

6 Conclusions

The purpose of this work has been to develop and experiment a new dynamic programming algorithm to solve the $0-1$ multi-objective knapsack problem. We showed that by using several complementary dominance relations, we obtain a method which outperforms experimentally the existing methods. In addition, our method is extremely efficient with regard to the other methods on the conflicting instances that model real-world applications. Lastly, this method is the first one to our knowledge that can be applied for knapsack problems with more than two objectives and the results in the three-objective case are satisfactory.

While we focused in this paper on the $0-1$ multi-objective knapsack problem, we could envisage in future research to apply dominance relations based on similar ideas to other multi-objective problems, admitting a direct dynamic programming formulation, such as the multi-objective shortest path problem or some multi-objective scheduling problems.

References

[1] M. Ehrgott and X. Gandibleux. A survey and annoted bibliography of multiobjective combinatorial optimization. *OR Spektrum*, 22(4):425–460, 2000.

[2] M. Ehrgott. *Multicriteria optimization*. LNEMS 491. Springer, Berlin, 2005.

[3] S. Martello and P. Toth. *Knapsack Problems*. Wiley, New York, 1990.

[4] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, Berlin, 2004.

[5] Meir J. Rosenblatt and Zilla Sinuany-Stern. Generating the discrete efficient frontier to the capital budgeting problem. *Operations Research*, 37(3):384–394, 1989.

[6] Juann-Yuan Teng and Gwo-Hshiung Tzeng. A multiobjective programming approach for selecting non-independent transportation investment alternatives. *Transportation Research-B*, 30(4):201–307, 1996.

[7] M. M. Kostreva, W. Ogryczak, and D. W. Tonkyn. Relocation problems arising in conservation biology. *Computers and Mathematics with Applications*, 37(4-5):135–150, 1999.

[8] Larry Jenkins. A bicriteria knapsack program for planning remediation of contaminated lightstation sites. *European Journal of Operational Research*, 140(2):427–433, 2002.

[9] K. Klamroth and M. Wiecek. Dynamic programming approaches to the multiple criteria knapsack problem. *Naval Research Logistics*, 47(1):57–76, 2000.

[10] 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(2):139–155, 1998.

[11] M. E. Captivo, J. Climaco, J. Figueira, E. Martins, and J. L. Santos. Solving bicriteria 0-1 knapsack problems using a labeling algorithm. *Computers and Operations Research*, 30(12):1865–1886, 2003.

[12] C. Silva, J. Clímaco, and J. Figueira. Core problems in the bi-criteria {0,1} knapsack: new developments. Research Report 12, INESC-Coimbra, 2005.

[13] T. Erlebach, H. Kellerer, and U. Pferschy. Approximating multiobjective knapsack problems. *Management Science*, 48(12):1603–1612, 2002.

[14] X. Gandibleux and A. Freville. Tabu search based procedure for solving the $0-1$ multiobjective knapsack problem: the two objectives case. *Journal of Heuristics*, 6(3):361–383, 2000.

[15] C.G. Da Silva, J. Climaco, and J. Figueira. A scatter search method for bi-criteria {0-1}-knapsack problems. *European Journal of Operational Research*, 169(2):373–391, 2006.

[16] C.G. Da Silva, J. Climaco, and J. Figueira. Integrating partial optimization with scatter search for solving bi-criteria {0-1}-knapsack problems. *European Journal of Operational Research*, 177(3):1656–1677, 2007.

[17] C. Berge. *Graphs*. North Holland, 1985.

[18] H.M. Weignartner and D.N. Ness. Methods for the solution of the multi-dimensional 0/1 knapsack problem. *Operations Research*, 15(1):83–103, 1967.

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

[20] M. Ehrgott and X. Gandibleux. Bound sets for biobjective combinatorial optimization problems. *Computers and Operations Research*, 34(9):2674–2694, 2007.

[21] H.T. Kung, F. Luccio, and F.P. Preparata. On finding the maxima of set of vectors. *Journal of the Association for Computing Machinery*, 22(4):469–476, 1975.

[22] V. Chankong and Y. Y. Haimes. *Multiobjective decision making*. Elsevier Science Publishing, New York, 1983.