# An efficient implementation for 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. {bazgan,hugot,vdp}@lamsade.dauphine.fr

**Abstract.** In this paper, we present an approach, based on dynamic programming, for solving 0-1 multi-objective knapsack problems. 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, 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). The reader can refer to [1] about multi-objective combinatorial optimization.

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*, [2,3]). Moreover, in the multi-objective case, many real-world applications are reported dealing with capital budgeting [4], relocation issues arising in conservation biology [5], and planning remediation of contaminated lightstation sites [6].

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 [7], 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 [8], and the method of [9], based on transformation of the problem into a bi-objective shortest path problem. All these methods have been designed for the bi-objective case and cannot be extended in a straightforward way to a higher number of objectives.

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 of [9], 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 the dynamic programming approach and the 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. All proofs are given in the appendix 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 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 $\Delta$ 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* 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 $b \succsim a$, and $B \subseteq A$ 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 $v_1^k, \ldots, v_p^k$ $(k = 1, \ldots, n)$. A feasible solution is represented by a vector $x = (x_1, \ldots, x_n)$ 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 $i$th objective is $f_i(x) = \sum_{k=1}^{n} v_i^k x_k$ $(i = 1, \ldots, p)$. 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 dominance relations in DP (section 3.2).

### 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 $s^k = (s_1^k, \ldots, s_p^k, s_{p+1}^k) \in S^k$ represents a feasible solution of value $s_i^k$ on the $i$th objective $(i = 1, \ldots, p)$ and of weight $s_{p+1}^k$. Thus, we have $S^k = S^{k-1} \cup \{(s_1^{k-1} + v_1^k, \ldots, s_p^{k-1} + v_p^k, s_{p+1}^{k-1} + w^k) : s_{p+1}^{k-1} + w^k \leq W, s^{k-1} \in S^{k-1}\}$ 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. Thus, 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 $s_i^k \geq \tilde{s}_i^k$, $i = 1, \ldots, p$.

**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 $s_{p+1}^k + \sum_{j \in J} w^j \leq W$. 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 $s_i^n = s_i^k + \sum_{j \in J} v_i^j$ for $i = 1, \ldots, p$ and $s_{p+1}^n = s_{p+1}^k + \sum_{j \in J} w^j$. 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. Dominance relations are 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 $s^k, \tilde{s}^k \in S^k$,*

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

A dominance relation $D^k$ is not necessarily transitive. However, due to the transitivity of $\underline{\Delta}$, if $D^k$ is a dominance relation then its transitive closure $\widehat{D}^k$ is also a dominance relation.

We introduce now a way of using dominance relations in Algorithm DP (see Algorithm 1). At each phase $k$, Algorithm DP generates a subset of states $C^k \subseteq S^k$. This is achieved by first creating from $C^{k-1}$ a temporary subset $T^k \subseteq S^k$. Then, we apply dominance relation $D^k$ to each state of $T^k$ in order to check if it is not dominated by any state already in $C^k$ (in which case it is added to $C^k$) and if it dominates states already in $C^k$ (which are then removed from $C^k$).

**Algorithm 1**: Dynamic Programming

```
1  $C^0 \leftarrow \{(0, \ldots, 0)\}$;
2  for $k \leftarrow 1$ to $n$ do
3      $T^k \leftarrow C^{k-1} \cup \{(s_1^{k-1} + v_1^k, \ldots, s_p^{k-1} + v_p^k, s_{p+1}^{k-1} + w^k) | s_{p+1}^{k-1} + w^k \leq W, s^{k-1} \in C^{k-1}\}$;
       /* Assume that $T^k = \{s^{k(1)}, \ldots, s^{k(r)}\}$ */
4      $C^k \leftarrow \{s^{k(1)}\}$;
5      for $i \leftarrow 2$ to $r$ do
           /* Assume that $C^k = \{\tilde{s}^{k(1)}, \ldots, \tilde{s}^{k(\ell_i)}\}$ */
6          nonDominated $\leftarrow$ true ; $j \leftarrow 1$;
7          while $j \leq \ell_i$ and nonDominated do
8              if $\tilde{s}^{k(j)} D^k s^{k(i)}$ then nonDominated $\leftarrow$ false
9              else if $s^{k(i)} D^k \tilde{s}^{k(j)}$ then $C^k \leftarrow C^k \backslash \{\tilde{s}^{k(j)}\}$;
10             $j \leftarrow j + 1$;
11         while $j \leq \ell_i$ do
12             if $s^{k(i)} D^k \tilde{s}^{k(j)}$ then $C^k \leftarrow C^k \backslash \{\tilde{s}^{k(j)}\}$;
13             $j \leftarrow j + 1$;
14         if nonDominated then $C^k \leftarrow C^k \cup \{s^{k(i)}\}$;
15 return $C^n$;
```

The following results characterize the set $C^k$ obtained at the end of each phase $k$ and establish the validity of Algorithm DP.

**Proposition 1.** *For any dominance relation $D^k$ on $S^k$, the set $C^k$ obtained at the end of phase $k$ in Algorithm DP is a covering set of $T^k$ with respect to $\widehat{D}^k$ that is also independent with respect to $D^k$ ($k = 1, \ldots, n$).*

*Proof.* Clearly, $C^k$ is independent with respect to $D^k$, since we insert in $C^k$ a state $s^k$ at step 14 only if it is non-dominated by all others states of $C^k$ (step 8) and we have removed from $C^k$ all states dominated by $s^k$ (step 9).

We have $\tilde{s}^k \in T^k \backslash C^k$ either because it did not pass the test at step 8 or was removed at step 9 or 12. In both cases, this is due to a state $\bar{s}^k$ already in $C^k$ or to be included in $C^k$ (at step 14) such that $\bar{s}^k \widehat{D}^k \tilde{s}^k$. Indeed, in the first case this is obvious since we have $\bar{s}^k D^k \tilde{s}^k$, and in the second case we can have either $\bar{s}^k D^k \tilde{s}^k$ or there exists a state $\bar{s}'^k$ such that $\bar{s}'^k D^k \tilde{s}^k$, that is not added to $C^k$ (step 14) due to a state $\bar{s}^k$ currently in $C^k$ (step 8) such that $\bar{s}^k D^k \bar{s}'^k D^k \tilde{s}^k$. In both cases, it may happen that $\bar{s}^k$ will be removed from $C^k$ at a later iteration of the for loop (at step 9 or 12) if there exists a new state $\hat{s}^k \in T^k$, such that $\hat{s}^k D^k \bar{s}^k$. However, transitivity of $\widehat{D}^k$ ensures the existence, at the end of phase $k$, of a state $s^k \in C^k$ such that $s^k \widehat{D}^k \tilde{s}^k$. □

**Theorem 1.** *For any family of dominance relations $D^1, \ldots, D^n$, Algorithm DP returns $C^n$ a covering set of $S^n$ with respect to $\underline{\Delta}$. Moreover, if $D^n = \underline{\Delta}$, $C^n$ 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 using $D^k$ during phases $k \leq n$. Let $k_1$ be the highest phase where $T^{k_1}$ still contains restrictions of $\tilde{s}^n$, which will be removed by applying $D^{k_1}$. Consider any of these restrictions, denoted by $\tilde{s}^{k_1}_{(n)}$. Since $\tilde{s}^{k_1}_{(n)} \in T^{k_1} \backslash C^{k_1}$, we know from Proposition 1, that there exists $s^{k_1} \in C^{k_1}$ such that $s^{k_1} \widehat{D}^{k_1} \tilde{s}^{k_1}_{(n)}$. Since $\widehat{D}^{k_1}$ is a dominance relation, by (1), we have that for all extensions of $\tilde{s}^{k_1}_{(n)}$, and in particular for $\tilde{s}^n$, there exists $s^{n_1} \in \text{Ext}(s^{k_1})$ 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 $T^{k_2}$ still contains restrictions of $s^{n_1}$, which will be removed by applying $D^{k_2}$. Consider any of these restrictions, denoted by $s^{k_2}_{(n_1)}$. As before, we establish the existence of a state $s^{k_2} \in C^{k_2}$ such that there exists $s^{n_2} \in \text{Ext}(s^{k_2})$ 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 $s^n \underline{\Delta} \tilde{s}^n$.

By Proposition 1, if $D^n = \underline{\Delta}$, $C^n$ is an independent set with respect to $\underline{\Delta}$. Thus $C^n$, which corresponds to a reduced efficient set, represents the set of non dominated vectors. □

When dominance relation $D^k$ is transitive, Algorithm DP can be drastically simplified in several ways. First, when we identify, at step 8, a state $\tilde{s}^{k(j)} \in C^k$ such that $\tilde{s}^{k(j)} D^k s^{k(i)}$, transitivity of $D^k$ and independence of $C^k$ with respect to $D^k$ ensure that $s^{k(i)}$ cannot dominate any state in $C^k$, which makes the loop 11-13 useless. Second, if we identify, at step 9, a state $\tilde{s}^{k(j)} \in C^k$ such that $s^{k(i)} D^k \tilde{s}^{k(j)}$, transitivity of $D^k$ and independence of $C^k$ with respect to $D^k$ ensure that $s^{k(i)}$ cannot be dominated by a state of $C^k$, which allows us to leave immediately the current loop 7-10.

Further improvements can still be made since it is usually possible to generate states of $T^k = \{s^{k(1)}, \ldots, s^{k(r)}\}$ according to a *dominance preserving order* for $D^k$ such that for all $i < j$ $(1 \leq i,j \leq r)$ we have either not$(s^{k(j)} D^k s^{k(i)})$ or $\left(s^{k(j)} D^k s^{k(i)} \text{ and } s^{k(i)} D^k s^{k(j)}\right)$. The following proposition gives a necessary and sufficient condition to establish the existence of a dominance preserving order for a dominance relation.

**Proposition 2.** *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 $s^{k(i)}$ on this cycle with $j > i$, a contradiction.
$\Leftarrow$ Any topological order based on the asymmetric part of $D^k$ is a dominance preserving order for $D^k$. $\square$

If states of $T^k$ are generated according to a dominance preserving order for $D^k$, step 9 and loop 11-13 can be omitted.

In our presentation, Algorithm DP provides us with the set of non-dominated criterion vectors. The approach can be easily adapted to obtain a reduced efficient set by adding to each generated state components characterizing its corresponding solution. Moreover, the efficient set can be obtained by using dominance relations $D^k$ $(k = 1, \dots, n)$ satisfying condition (1), where $\underline{\Delta}$ is replaced by $\Delta$, and provided that $C^n$ is an independent set with respect to $\Delta$.

## 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 a brief explanation of 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) [2,3]. For the multi-objective version, there is no such a natural order.

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

In the computational experiments, in Section 5, 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 $J = \{k+1,\ldots,n\}$. It is then 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:

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

**Proposition 3 (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 2, since $D_r^k$ is transitive. □

This dominance relation is quite 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.

A second dominance relation $D_{\underline{\Delta}}^k$ is defined on $S^k$ for $k = 1,\ldots,n$ by:

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

Dominance relation $D_{\underline{\Delta}}^k$ is a generalization to the multi-objective case of the dominance relation usually attributed to Weingartner and Ness [10] and used in the classical Nemhauser and Ullmann algorithm [11].

**Proposition 4 (Relation $D^k_{\underline{\Delta}}$).**

*(a) $D^k_{\underline{\Delta}}$ is a dominance relation* *(c) $D^k_{\underline{\Delta}}$ admits dominance preserving orders*
*(b) $D^k_{\underline{\Delta}}$ is transitive* *(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 2, 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 extensions of a state and an upper bound of the extensions of another state. In our context, a 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 $s^n \in \text{Ext}(s^k)$ 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 $J''$ defined as follows. After relabeling items $k+1, \ldots, n$ according to order $\mathcal{O}^{sum}$, completion $J'$ is obtained by inserting sequentially the remaining items into the solution provided that the capacity constraint is respected. More precisely, $J'$ correspond to $J_n$ where $J_k = \emptyset$ and $J_\ell = J_{\ell-1} \cup \{\ell\}$ if $s^k_{p+1} + \sum_{j \in J_{\ell-1}} w^j + w^\ell \leq W$, $\ell = k+1, \ldots, n$. $J''$ is defined similarly by relabeling items according to order $\mathcal{O}^{\max}$.

To compute $u$, we use the upper bound presented in [2] 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 [2, Th 2.2], when items $k+1, \ldots, n$ are relabeled according to order $\mathcal{O}^i$, an upper bound on the $i$th criterion value of $s^k \in S^k$ is for $i = 1, \ldots, p$:

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

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

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

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

**Proposition 5 (Relation $D_b^k$).**
*(a) $D_b^k$ is a dominance relation* *(c) $D_b^k$ admits dominance preserving orders*
*(b) $D_b^k$ is transitive* *(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 $s_i^k + \sum_{j \in J_1} v_i^j \geq \tilde{u}_i$ ($i = 1, \ldots, p$), and on the other hand, there exists $J_2 \in \{J', J''\}$ such that $\tilde{s}_i^k + \sum_{j \in J_2} v_i^j \geq \bar{u}_i$ ($i = 1, \ldots, p$). Since $\tilde{u}$ is an upper bound for $\tilde{s}^k$ we have $\tilde{u}_i \geq \tilde{s}_i^k + \sum_{j \in J_2} v_i^j$ ($i = 1, \ldots, p$). Thus we get $s^k D_b^k \bar{s}^k$.
(c) By Proposition 2, 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 [12]. 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 Use of 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 $D_b^k$).

The running time of Algorithm DP using these relations is in $O(n(\min\{W, U\}$ $U^{p-1})^2)$ where $U$ is an upper bound on the value of all solutions on all criteria, since $C^k$, which contains only non-dominated vectors with respect to profit values and weight, has a cardinality in $O(\min\{W, U\}U^{p-1})$. Based on ideas of [13], in the bi-objective case, in order to remove efficiently dominated states at each phase, we use an AVL tree [14, sec. 6.3.3] for storing states which leads to a significant improvement of the running time to $O(n \min\{W, U\} \log(\min\{W, U\}))$. Observe that space complexity of Algorithm DP is in $O(\min\{W, U\}U^{p-1})$.

## 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, $v_k^1 \in_R [1, 1000]$, $v_k^2 \in_R [1, 1000]$ and $w_k \in_R [1, 1000]$

**B:** Unconflicting instances, where $v_k^1$ is correlated with $v_k^2$, *i.e.* $v_k^1 \in_R [111, 1000]$ and $v_k^2 \in_R [v_k^1 - 100, v_k^1 + 100]$, and $w_k \in_R [1, 1000]$

**C:** Uncorrelated conflicting instances, where $v_k^1$ and $v_k^2$ are mirror values, *i.e.* $v_k^1 \in_R [1, 1000]$, $v_k^2 \in_R [\max\{900 - v_1^k; 1\}, \min\{1100 - v_1^k; 1000\}]$, and $w_k \in_R [1, 1000]$

**D:** Correlated conflicting instances, where $v_k^1$ and $v_2^k$ are mirror values, and $w_k$ is correlated with $v_k^1$ and $v_2^k$, *i.e.* $v_k^1 \in_R [1, 1000]$, $v_k^2 \in_R [\max\{900 - v_1^k; 1\}, \min\{1100 - v_1^k; 1000\}]$, 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 made on instances of type A. Sometimes, other instances such as those of type B, which were introduced in [9], are studied. However, instances of type B should be viewed as quasi mono-criterion instances since they involve two non conflicting criteria. 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. For instances of types C and D, items are around the line $y = -x + 1000$. In instances of type D, $w^k$ is correlated with $v_k^1, v_k^2$. These instances were introduced in order to verify if correlated instances are harder than uncorrelated instances as in the single-criterion context [2].

For three-objective experiments, we considered the generalization of random instances of type A where $v_k^i \in_R [1, 1000]$ for $i = 1, \ldots, 3$ 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.

## 5.2 Results in the bi-objective case

First, in the experiments, we try to determine the best order to sort items for DP. Table 1 shows clearly that the way of ordering items has a dramatic impact on the CPU time and that order $\mathcal{O}^{\max}$ is significantly better for all types of instances. Thus, in the following, items are sorted and labeled according to $\mathcal{O}^{\max}$.

**Table 1.** Impact of different orders of items in our approach (Average CPU time in seconds, $p = 2$).

| Type | n | $\mathcal{O}^{\max}$ | $\mathcal{O}^{sum}$ | $\mathcal{O}^{\min}$ | Random |
|---|---|---|---|---|---|
| A | 300 | 84.001 | 100.280 | 94.598 | 178.722 |
| B | 600 | 1.141 | 1.084 | 1.403 | 77.699 |
| C | 200 | 59.986 | 60.061 | 85.851 | 107.973 |
| D | 90 | 20.795 | 23.687 | 35.426 | 31.659 |

Second, we show the complementarity of dominance relations $D_r^k$, $D_{\underline{\Delta}}^k$, and $D_b^k$. Table 2 establishes that it is always better to use the three relations, due to their complementarity.

**Table 2.** Complementarity of dominance relations $D_r^k$, $D_{\underline{\Delta}}^k$, and $D_b^k$ in our approach (Average CPU time in seconds, $p = 2$).

| Type | n | $D_{\underline{\Delta}}^k$ | $D_r^k$ and $D_{\underline{\Delta}}^k$ | $D_{\underline{\Delta}}^k$ and $D_b^k$ | $D_r^k$, $D_{\underline{\Delta}}^k$, and $D_b^k$ |
|---|---|---|---|---|---|
| A | 300 | 272.628 | 157.139 | 85.076 | 84.001 |
| B | 600 | 230.908 | 174.015 | 1.188 | 1.141 |
| C | 200 | 122.706 | 63.557 | 61.696 | 59.986 |
| D | 90 | 46.137 | 24.314 | 23.820 | 20.795 |

Lastly, we present, in Table 3, the performance of our approach on large size instances. 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. As predicted, instances of type B are quasi mono-objective instances and have very few non-dominated criterion vectors. 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 3.** Results of our approach on large size instances ($p = 2$).

| 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 |
| | 300 | 57.475 | 84.001 | 101.354 | 905 | 1130.7 | 1651 | 898524.7 |
| | 500 | 677.398 | 889.347 | 1198.190 | 2034 | 2537.5 | 2997 | 5120514.7 |
| | 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 |
| | 300 | 311.995 | 373.097 | 470.429 | 2510 | 2893.6 | 3297 | 3481238.4 |
| | 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 |

### 5.3 Comparison with other exact methods in the bi-objective case

The results of a comparative study, in the bi-objective case, between the exact method of [9], an exact method based on a commercial Integer Programming (IP) solver and our approach using $D_r^k$, $D_{\underline{\Delta}}^k$, and $D_b^k$ are presented in Table 4. We have selected the method of [9] since it is the most efficient method currently known. An exact method based on a commercial IP solver has been selected, on one hand, because it is relatively easy to implement, and on the other hand, since each efficient solution is found by solving only one linear program, this method has much less storage problems than the two others.

An exact method based on a commercial IP solver is presented in Algorithm 2. This algorithm relies on the idea that since the decision space $Z = \{f(x) : x \in X\}$ is included in $\mathbb{N}^2$, all efficient solutions can be enumerated in decreasing order of value on the first criterion. Cplex 9.0 is used as IP solver in Algorithm 2 which is written in C++.

**Algorithm 2**: Computing a reduced efficient set with an IP Solver

```
1 Generate $y$ an optimal solution of $\max_{x \in X} f_1(x)$ and $z$ an optimal solution of $\max_{x \in X} f_2(x)$;
2 Generate $x^1$ an 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
5     $\alpha \leftarrow f_2(z) - f_2(x^j) - 1$;
6     Generate $x^{j+1}$ an optimal solution of $\max\{\alpha f_1(x) + f_2(x) : x \in X, f_2(x) \geq f_2(x^j) + 1\}$;
7     $X^\star \leftarrow X^\star \cup \{x^{j+1}\}$ ; $j \leftarrow j + 1$;
8 return $X^\star$;
```

The three methods have been used on the same instances and the same computer. For the exact method of [9], we used the source code, in C, obtained from the authors. Table 4 presents results, in the bi-objective case, for instances of type A, B, C, and D for increasing size of $n$ while the method of [9] can solve all instances of the series considered. Since the method of [9] is very storage consuming, it can only solve instances of type A up to 300 items, of type B up to 800 items, of type C up to 100 items and of type D up to 100 items whereas we recall (see Table 3) that our approach can solve instances respectively up to 700, 4000, 500 and 250 items.

**Table 4.** Comparison between the exact method of [9], Algorithm 2 using Cplex and our approach.

| Type | $n$ | Avg time in (s) | | | Avg |
|---|---|---|---|---|---|
| | | [9] | Cplex | Our approach | $\|ND\|$ |
| A | 100 | 2.476 | 5.343 | 0.328 | 159.3 |
| | 200 | 37.745 | 57.722 | 12.065 | 529.0 |
| | 300 | 163.787 | 285.406 | 84.001 | 1130.7 |
| B | 600 | 27.694 | 27.543 | 1.141 | 74.3 |
| | 700 | 47.527 | 29.701 | 2.299 | 78.6 |
| | 800 | 75.384 | 68.453 | 5.280 | 118.1 |
| C | 100 | 12.763 | 208.936 | 2.869 | 558.2 |
| D | 100 | 127.911 | 23126.926 | 40.866 | 1765.4 |

Considering CPU time, we can conclude that our approach is always faster than the exact method of [9] and than Algorithm 2 with Cplex on the considered instances. We can also observe that the CPU time needed to solve correlated and conflicting instances of type D by Algorithm 2 with Cplex is especially large (about 6.5 hours in average for instances 2D100). In addition, we can remark that the exact method of [9] cannot solve conflicting instances (type C and D) of moderate and large size for which the number of non-dominated criterion vectors is large. Indeed, the exact method of [9] does not work very well on instances with many non-dominated criterion vectors due to storage limitations.

## 5.4 Results in the three-objective case

In table 5, we present results of our approach concerning large size instances of type $A$ 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 5.** Results of our approach on instances of type $A$ in the three-objective case.

| $n$ | Time in (s) | | | $\|ND\|$ | | | Avg |
|---|---|---|---|---|---|---|---|
| | Min | Avg | Max | Min | Avg | Max | $\max_k\{\|C^k\|\}$ |
| 10 | 0.000 | 0.000 | 0.000 | 4 | 8.3 | 18 | 20.9 |
| 30 | 0.000 | 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 |

## 6 Conclusions

The goal 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 with more than two objectives and the results in the three-objective case are very 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 such as the multi-objective shortest path problem or multi-objective scheduling problems.

## References

1. Ehrgott, M.: Multicriteria optimization. LNEMS 491. Springer, Berlin (2005)
2. Martello, S., Toth, P.: Knapsack Problems. Wiley, New York (1990)
3. Kellerer, H., Pferschy, U., Pisinger, D.: Knapsack Problems. Springer (2004)
4. Rosenblatt, M.J., Sinuany-Stern, Z.: Generating the discrete efficient frontier to the capital budgeting problem. Operations Research **37**(3) (1989) 384–394
5. Kostreva, M.M., Ogryczak, W., Tonkyn, D.W.: Relocation problems arising in conservation biology. Comp. and Math. with App. **37** (1999) 135–150
6. Jenkins, L.: A bicriteria knapsack program for planning remediation of contaminated lightstation sites. Eur. J. Oper. Res. **140** (2002) 427–433
7. Klamroth, K., Wiecek, M.: Dynamic programming approaches to the multiple criteria knapsack problem. Naval Research Logistics **47**(1) (2000) 57–76
8. Visée, M., Teghem, J., Pirlot, M., Ulungu, E.: Two-phases method and branch and bound procedures to solve the bi-objective knapsack problem. Journal of Global Optimization **12** (1998) 139–155
9. Captivo, M.E., Climaco, J., Figueira, J., Martins, E., Santos, J.L.: Solving bicriteria 0-1 knapsack problems using a labeling algorithm. Computers and Operations Research **30** (2003) 1865–1886
10. Weignartner, H., Ness, D.: Methods for the solution of the multi-dimensional 0/1 knapsack problem. Operations Research **15**(1) (1967) 83–103
11. Nemhauser, G., Ullmann, Z.: Discrete dynamic programming and capital allocation. Management Science **15**(9) (1969) 494–505
12. Ehrgott, M., Gandibleux, X.: Bound sets for biobjective combinatorial optimization problems. To appear in Computers and Operations Research (2007)
13. Kung, H., Luccio, F., Preparata, F.: On finding the maxima of set of vectors. J. Assoc. Comput. Mach. **22**(4) (1975) 469–476
14. Knuth, D.E.: The Art of Computer Programming, Vol. 3. Addison Wesley (1997)