# Implementing an efficient fptas for the 0–1 multi-objective knapsack problem

Cristina Bazgan, Hadrien Hugot, and Daniel Vanderpooten

*Université Paris-Dauphine, LAMSADE*

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

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

**Abstract**

In the present work we are interested in the practical behavior of a new fptas to solve the approximation version of the 0-1 multi-objective knapsack problem. The proposed methodology makes use of very general techniques (such as dominance relations in dynamic programming) and thus may be applicable in the implementation of fptas for other problems as well.

Extensive numerical experiments on various types of instances in the bi and tri-objective cases establish that our method performs very well both in terms of CPU time and size of solved instances. We point out some reasons for the good practical performance of our algorithm. A comparison with an exact method and the fptas proposed in Erlebach et al. (2002) is also performed.

*Keywords:* multi-objective knapsack problem, approximation, dynamic programming, dominance relations, combinatorial optimization.

## 1 Introduction

In multi-objective combinatorial optimization, a major challenge is to generate either the set of efficient solutions, that have the property that no improvement on any objective is possible without sacrificing on at least another objective, or the set of non-dominated criterion vectors corresponding to their image in the criterion space. The reader can refer to Ehrgott (2005) about multi-objective combinatorial optimization. However, even for moderately-sized problems, it is usually computationally prohibitive to identify the efficient set for two major reasons. First, the number of efficient solutions can be very large. This occurs notably when solving *intractable* instances of combinatorial multi-objective problems, for which the number of efficient solutions is not polynomial in the size of these instances (see, e.g., Ehrgott (2005) about the intractability of multi-objective problems). Second, for most multi-objective problems, deciding whether a given solution is dominated is $NP$-hard, even if the underlying single-objective problem can be solved in a polynomial time (see, e.g., Serafini (1986) about the $NP$-hardness of multi-objective problems).

To handle these two difficulties, researchers have been interested in developing approximation algorithms with provable a priori guarantee such as fully polynomial time approximation schemes (fptas). Indeed, an fptas computes, for a given accuracy $\varepsilon > 0$, in a running time that is polynomial both in the size of the input and in $1/\varepsilon$, an $(1+\varepsilon)$-approximation, that is a subset of solutions which contains, for each efficient solution, a solution that is at most at a factor $(1+\varepsilon)$ on all objective values. This is made possible since it has been pointed out in Papadimitriou and Yannakakis (2000) that, under certain general assumptions, there always exists an $(1+\varepsilon)$-approximation, with any given accuracy $\varepsilon > 0$, whose size is polynomial both in the size of the instance and in $1/\varepsilon$. Thus using an fptas for solving a multi-objective problem has two main advantages: on the one hand it provides us with an efficient algorithm to compute an approximation with a guaranteed accuracy and on the other hand it computes an approximation of reasonable size. Nevertheless, in this stream, researchers are usually motivated by the theoretical question of proving or disproving the existence of an fptas for a given problem (Warburton, 1987; Erlebach et al., 2002) or for a class of problems (Safer and Orlin, 1995a,b; Papadimitriou and Yannakakis, 2000; Angel et al., 2003). Thus, practical implementations of fptas are cruelly lacking and most of the schemes proposed in the literature are not efficient in practice.

We consider in this paper the 0–1 multi-objective knapsack problem which has been shown to admit an fptas in Safer and Orlin (1995a,b) and in Erlebach et al. (2002). Our perspective, however, is to propose another fptas focusing on its practical behavior. The main idea of our approach, based on dynamic programming, relies on the use of several complementary dominance relations to discard partial solutions. In a previous work (Bazgan et al., 2009), such techniques have been proved to be extremely efficient to solve the exact version of this problem. Extensive numerical experiments on various types of instances in the bi and tri-objective cases are reported and establish that our method performs very well both in terms of CPU time and size of solved instances (up to 20 000 items in less than 1 hour in the bi-objective case). We compare our approach with the exact method of Bazgan et al. (2009), which is the most effective exact method currently known and with the fptas proposed in Erlebach et al. (2002). In our experiments, we point out some reasons for the good practical performance of our algorithm that may be applicable to other fptas. Indeed, since our methodology relies on very general techniques (such as dominance relations in dynamic programming), it may be applicable in the implementation of fptas for other problems as well.

This paper is organized as follows. In Section 2, we review basic concepts about multi-objective optimization and approximation, and formally define the 0–1 multi-objective knapsack problem. Section 3 presents the dynamic programming approach using dominance relations. Section 4 is devoted to the presentation of the dominance relations. Computational experiments and results are reported in Section 5. Conclusions are provided in a final section.

## 2 Preliminaries

We first 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 $b, b' \in B,\ b \neq b',\ \text{not}(b \succsim b')$.

### 2.1 Multi-objective optimization and approximation

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*. The set of non-dominated criterion vectors is denoted by $ND$. A set of efficient solutions is said to be *reduced* if it contains only one solution corresponding to each non-dominated criterion vector. Observe that $X' \subseteq X$ is a reduced efficient set if and only if it is a covering and independent set with respect to $\underline{\Delta}$.

For any constant $\varepsilon \geq 0$, the relation $\underline{\Delta}_\varepsilon$, called $\varepsilon$-dominance, defined on $X$, states that for all $x, x' \in X,\ x \underline{\Delta}_\varepsilon x'$ if and only if $f_i(x)(1+\varepsilon) \geq f_i(x')$ for $i = 1, \dots, p$. For any constant $\varepsilon \geq 0$, an $(1+\varepsilon)$-*approximation* is a covering set of $X$ with respect to $\underline{\Delta}_\varepsilon$. Any $(1+\varepsilon)$-approximation which does not contain solutions that dominate each other, *i.e.* which is independent with respect to $\underline{\Delta}$, is *a reduced* $(1+\varepsilon)$*-approximation.* In the following, $ND_\varepsilon$ denotes the image in the criterion space of a reduced $(1+\varepsilon)$-approximation.

### 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 capacity 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 consider two versions: *the exact version* which aims at determining a reduced efficient set, and *the approximation version* which aims at determining a reduced $(1+\varepsilon)$-approximation. Several dynamic programming formulations have been proposed in Klamroth and Wiecek (2000) for the exact version. We focus now on the approximation version.

## 3 Dynamic Programming for the approximation version

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 for solving the approximation version by a DP approach (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 a corresponding feasible solution. Thus, relations defined on $X$ are also valid on $S^k$, and we have $s^k \underline{\Delta} \tilde{s}^k$ if and only if $s_i^k \geq \tilde{s}_i^k$, $i = 1, \ldots, p$ and $s^k \underline{\Delta}_\varepsilon \tilde{s}^k$ if and only if $s_i^k(1+\varepsilon) \geq \tilde{s}_i^k$, $i = 1, \ldots, p$

**Definition 1 (Completion, extension, restriction)** *For any state $s^k \in S^k$ $(k \leq n)$, 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 Families of dominance relations in Dynamic Programming

The efficiency of DP depends crucially on the possibility of reducing the set of states at each phase. In the context of the approximation version, a family of dominance relations between states for $\underline{\Delta}_\varepsilon$ is used to discard states at any phase. Each dominance relation of this family is specific to a phase. Indeed, we share out the total error $\varepsilon$ between the phases by the mean of an error function and associate to each dominance relation of the family a proportion of this error.

**Definition 2 (Error function)** *The function $e : \{1, \ldots, n\} \to \mathbb{R}$ is an error function if and only if $\sum_{k=1}^{n} e(k) \leq 1$ and $e(k) \geq 0,\ k = 1, \ldots, n$.*

Families of dominance relations between states for $\underline{\Delta}_\varepsilon$ can then be defined as follows.

**Definition 3 (Families of dominance relations between states for $\underline{\Delta}_\varepsilon$)** *For any $\varepsilon \geq 0$ and any error function $e$, a family of relations $D^k$ on $S^k$, $k = 1, \ldots, n$, is a family of dominance relations for $\underline{\Delta}_\varepsilon$ if for all $s^k, \tilde{s}^k \in S^k$,*

$$s^k D^k \tilde{s}^k \Rightarrow \forall \tilde{s}^n \in Ext(\tilde{s}^k), \exists s^n \in Ext(s^k), s_i^n (1+\varepsilon)^{e(k)} \geq \tilde{s}_i^n, \quad i = 1, \ldots, p \tag{1}$$

When $\varepsilon = 0$, Definition 3 collapses to the classical definition of dominance relations used in the context of the exact version:

**Definition 4 (Dominance relation between states for $\underline{\Delta}$)** *A relation $D^k$ on $S^k$, $k = 1, \ldots, n$, is a dominance relation for $\underline{\Delta}$ if for all $s^k, \tilde{s}^k \in S^k$,*

$$s^k D^k \tilde{s}^k \Rightarrow \forall \tilde{s}^n \in Ext(\tilde{s}^k), \exists s^n \in Ext(s^k), s^n \underline{\Delta} \tilde{s}^n \tag{2}$$

Even if dominance relations can be non-transitive, in order to be efficient in the implementation, we consider only transitive dominance relations $D^k$, $k = 1, \ldots, n$. We introduce now the way of using families of transitive dominance relations for $\underline{\Delta}_\varepsilon$ (see 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 $T^k \subseteq S^k$. Then, we apply $D^k$ 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 $C^k$). Observe that due to the transitivity of $D^k$, a state $s^k \in T^k$ that dominates a state of $C^k$ (step 9) cannot be dominated by a state already in $C^k$ (step 8).

The following results characterize the set $C^k$ obtained at the end of each phase $k$ and establish the validity of Algorithm 1.

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

**Algorithm 1**: Computing a reduced $(1+\varepsilon)$-approximation

```
1  $C^0 \leftarrow \{(0,\dots,0)\}$;
2  for $k \leftarrow 1$ to $n$ do
3      $T^k \leftarrow C^{k-1} \cup \{(s_1^{k-1}+v_1^k,\dots,s_p^{k-1}+v_p^k, s_{p+1}^{k-1}+w^k) | s_{p+1}^{k-1}+w^k \le W, s^{k-1} \in C^{k-1}\}$;
       /* Assume that $T^k = \{s^{k(1)},\dots,s^{k(r)}\}$ */
4      $C^k \leftarrow \{s^{k(1)}\}$;
5      for $i \leftarrow 2$ to $r$ do
           /* Assume that $C^k = \{\tilde{s}^{k(1)},\dots,\tilde{s}^{k(\ell_i)}\}$ */
6          $dominated \leftarrow$ false ; $dominates \leftarrow$ false ; $j \leftarrow 1$;
7          while $j \le \ell_i$ and not(dominated) and not(dominates) do
8              if $\tilde{s}^{k(j)} D^k s^{k(i)}$ then $dominated \leftarrow$ true
9              else if $s^{k(i)} D^k \tilde{s}^{k(j)}$ then $C^k \leftarrow C^k \backslash \{\tilde{s}^{k(j)}\}$ ; $dominates \leftarrow$ true;
10             $j \leftarrow j+1$;
11         if not(dominated) then
12             while $j \le \ell_i$ do
13                 if $s^{k(i)} D^k \tilde{s}^{k(j)}$ then $C^k \leftarrow C^k \backslash \{\tilde{s}^{k(j)}\}$;
14                 $j \leftarrow j+1$;
15             $C^k \leftarrow C^k \cup \{s^{k(i)}\}$;
16 return $C^n$;
```

*Proof:* Clearly, $C^k$ is independent with respect to $D^k$, since we insert a state $s^k$ into $C^k$ at step 15 only if it is not dominated by any other state of $C^k$ (step 8) and all states dominated by $s^k$ have been removed from $C^k$ (steps 9 and 13).

We show now that $C^k$ is a covering set of $T^k$ with respect to $D^k$. Consider $\tilde{s}^k \in T^k \backslash C^k$. This occurs either because it did not pass the test at step 8 or was removed at step 9 or 13. This is due respectively to a state $\bar{s}^k$ already in $C^k$ or to be included in $C^k$ (at step 15) such that $\bar{s}^k D^k \tilde{s}^k$. 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 13) if there exists a new state $\hat{s}^k \in T^k$ to be included in $C^k$, such that $\hat{s}^k D^k \bar{s}^k$. However, transitivity of $D^k$ ensures the existence, at the end of phase $k$, of a state $s^k \in C^k$ such that $s^k D^k \tilde{s}^k$. □

**Theorem 1** *For any family of transitive dominance relations $D^1,\dots,D^n$ for $\underline{\Delta}_\varepsilon$, Algorithm 1 returns $C^n$ a covering set of $S^n$ with respect to $\underline{\Delta}_\varepsilon$. Moreover, if $\underline{\Delta} \subseteq D^n$, $C^n$ is a reduced $(1+\varepsilon)$-approximation.*

*Proof:* Consider $s^n \in S^n \backslash C^n$. Thus, all its restrictions have been removed during some phases $k \le n$, when selecting a covering set $C^k$. Let $k_1$ be the highest phase during which the last restriction of $s^n$, denoted by $s^{n(k_1)} \in T^{k_1}$ is removed from $C^{k_1}$. Then, Proposition 1 ensures the existence of $s^{k_1} \in C^{k_1}$ such that $s^{k_1} D^{k_1} s^{n(k_1)}$. By (1), for all extensions

of $s^{n(k_1)}$, and in particular for $s^n$, there exists $s^{n_1} \in \text{Ext}(s^{k_1})$ such that $s_i^n \leq (1+\varepsilon)^{e(k_1)} s_i^{n_1}$ $(i = 1, \ldots, p)$. It may happen that all restrictions of $s^{n_1}$ will be removed when selecting a covering set at a later phase. In this case, there exists a phase $k_2 > k_1$, corresponding to the highest phase during which the last restriction of $s^{n_1}$, denoted by $s^{n_1(k_2)}$ is removed from $C^{k_2}$. As before, we establish the existence of a state $s^{k_2} \in C^{k_2}$ such that $s^{k_2} D^{k_2} s^{n_1(k_2)}$ and of a state $s^{n_2} \in \text{Ext}(s^{k_2})$ such that $s_i^{n_1} \leq (1+\varepsilon)^{e(k_2)} s_i^{n_2}$ $(i = 1, \ldots, p)$. Thus, we have $s_i^n \leq (1+\varepsilon)^{e(k_1)+e(k_2)} s_i^{n_2}$ $(i = 1, \ldots, p)$. By repeating this process, we establish the existence of a state $s^{n_t} \in C^n$ $(t \leq n)$, such that $s_i^n \leq (1+\varepsilon)^{\sum_{j=1}^{t} e(k_j)} s_i^{n_t}$ $(i = 1, \ldots, p)$. Since $e(k) \geq 0$ $(k = 1, \ldots, n)$, we have $\sum_{j=1}^{t} e(k_j) \leq \sum_{j=1}^{n} e(j) \leq 1$, thus we get $s_i^n \leq (1+\varepsilon) s_i^{n_t}$ $(i = 1, \ldots, p)$. This establishes that $C^n$ is an $(1+\varepsilon)$-approximation.
Moreover, if $\underline{\Delta} \subseteq D^n$, Proposition 1 ensures that $C^n$ is also independent with respect to $\underline{\Delta}$, which establishes that $C^n$ is a reduced $(1+\varepsilon)$-approximation. $\square$

Remark that when $\varepsilon = 0$, we have $\underline{\Delta}_\varepsilon = \underline{\Delta}$, and thus $C^n$ is a covering set of $X$ with respect to $\underline{\Delta}$. Moreover, in this case, if $\underline{\Delta} \subseteq D^n$, $C^n$ corresponds to a reduced efficient set.

## 4 Dominance relations

We first present a family of dominance relations for $\underline{\Delta}_\varepsilon$ that can provide an fptas in certain cases (Section 4.1). Then, we present two complementary dominance relations for $\underline{\Delta}$ (Section 4.2) and give a brief explanation of the way of applying them together with the family of dominance relations for $\underline{\Delta}_\varepsilon$ (Section 4.3).

### 4.1 Dominance relations for $\underline{\Delta}_\varepsilon$

Section 4.1.1 is devoted to the presentation of $D^k_{\underline{\Delta}_\varepsilon}$ $(k = 1, \ldots, n)$ a family of dominance relations for $\underline{\Delta}_\varepsilon$. In Section 4.1.2, we show that this family can provide an fptas in certain cases. In Section 4.1.3 we present different error functions that can be used in relations $D^k_{\underline{\Delta}_\varepsilon}$ $(k = 1, \ldots, n)$.

#### 4.1.1 A family of dominance relations for $\underline{\Delta}_\varepsilon$

We introduced in a previous work (Bazgan et al., 2009) a powerful dominance relation for $\underline{\Delta}$, in order to solve the exact version of the 0–1 multi-objective knapsack problem. This relation, denoted by $D^k_{\underline{\Delta}}$, is a generalization to the multi-objective case of the natural dominance relation, usually attributed to Weingartner and Ness (1967) and used in the classical Nemhauser and Ullmann algorithm (Nemhauser and Ullmann, 1969). Relation $D^k_{\underline{\Delta}}$ is defined on $S^k$ for $k = 1, \ldots, n$ by:

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

To solve the approximation version of the 0–1 multi-objective knapsack problem, we generalize relations $D^k_{\underline{\Delta}}$ ($k = 1, \ldots, n$), to obtain the family of dominance relations $D^k_{\underline{\Delta}_\varepsilon}$ ($k = 1, \ldots, n$) for $\underline{\Delta}_\varepsilon$ that is based on a partition of the criterion space into hyper-rectangles. For a constant $\varepsilon > 0$ and an error function $e$, we partition at each phase $k$ each positive criterion range $[1, U_i]$, where $U_i$ is an upper bound on the value of the feasible solutions on the $i$th criterion ($i = 1, \ldots, p$), into disjoint intervals of length $(1+\varepsilon)^{e(k)}$ ($k = 1, \ldots, n$). When $e(k) = 0$, we obtain the following degenerate intervals: $[1, 1], [2, 2], \ldots, [U_i, U_i]$ ($i = 1, \ldots, p$) and the number of intervals in the criterion space is in $O(U^p_{\max})$, where $U_{\max} = \max\{U_1, \ldots, U_p\}$. When $e(k) \neq 0$, we obtain the following intervals: $[1; (1+\varepsilon)^{e(k)}[, [(1+\varepsilon)^{e(k)}; (1+\varepsilon)^{2e(k)}[, \ldots, [(1+\varepsilon)^{(\ell^k_i-1)e(k)}; (1+\varepsilon)^{\ell^k_i e(k)}[$ where $\ell^k_i = \left\lfloor \frac{\log U_i}{e(k)\log(1+\varepsilon)} \right\rfloor + 1$ ($i = 1, \ldots, p$). Thus, the number of hyper-rectangles is in $O\Big(\big(\frac{\log U_{\max}}{e(k)\ \varepsilon}\big)^p\Big)$. In both cases, we add the interval $[0, 0]$. The number of the interval to which belongs the value of a state $s^k$ on the $i$th criterion ($i = 1, \ldots, p$) in this partition is then:

$$B_i(s^k, e(k)) = \begin{cases} s^k_i, \quad \text{if } e(k) = 0 \text{ or } s^k_i = 0 \\ \left\lfloor \frac{\log s^k_i}{e(k)\log(1+\varepsilon)} \right\rfloor + 1 \quad \text{otherwise} \end{cases}$$

From these partitions, we can define for any $\varepsilon > 0$ and any error function $e$, relations $D^k_{\underline{\Delta}_\varepsilon}$ on $S^k$ for $k = 1, \ldots, n$ by:

$$\text{for all } s^k, \tilde{s}^k \in S^k,\ s^k D^k_{\underline{\Delta}_\varepsilon} \tilde{s}^k \Leftrightarrow \begin{cases} B_i(s^k, e(k)) \geq B_i(\tilde{s}^k, e(k)) \quad i = 1, \ldots, p \quad \text{and} \\ s^k_{p+1} \leq \tilde{s}^k_{p+1} \text{ if } k < n \end{cases}$$

The following proposition shows that $D^k_{\underline{\Delta}_\varepsilon}$ is indeed a family of dominance relations for $\underline{\Delta}_\varepsilon$ and gives additional properties of $D^k_{\underline{\Delta}_\varepsilon}$.

**Proposition 2** *For any $\varepsilon > 0$ and any error function $e$, we have:*
*(a) $D^k_{\underline{\Delta}_\varepsilon}$, $k = 1, \ldots, n$, is a family of dominance relations for $\underline{\Delta}_\varepsilon$,*
*(b) for any $k \in \{1, \ldots, n\}$, $D^k_{\underline{\Delta}_\varepsilon}$ is transitive,*
*(c) for any $k \in \{1, \ldots, n\}$, $D^k_{\underline{\Delta}_\varepsilon} \supseteq D^k_{\underline{\Delta}}$ and $D^k_{\underline{\Delta}_\varepsilon} = D^k_{\underline{\Delta}}$ if $e(k) = 0$,*

*Proof:* (a) Consider two states $s^k$ and $\tilde{s}^k$ such that $s^k D^k_{\underline{\Delta}_\varepsilon} \tilde{s}^k$. This implies that $B_i(s^k, e(k)) \geq B_i(\tilde{s}^k, e(k))$ ($i = 1, \ldots, p$). If $k = n$, we get $s^n(1+\varepsilon)^{e(n)} \geq \tilde{s}^n$, which establishes condition (1) of Definition 3 for $k = n$. Otherwise, if $k < n$, 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 $\tilde{s}^n \in \text{Ext}(\tilde{s}^k)$, there exists $s^n \in \text{Ext}(s^k)$, based on the same completion as $\tilde{s}^n$, such that $s^n(1+\varepsilon)^{e(k)} \geq \tilde{s}^n$. This establishes that $D^k_{\underline{\Delta}_\varepsilon}$ satisfies condition (1) of Definition 3.
(b) Obvious.
(c) For any $k \in \{1, \ldots, n\}$, when $e(k) = 0$, we have by definition $D^k_{\underline{\Delta}_\varepsilon} = D^k_{\underline{\Delta}}$. For any $k \in \{1, \ldots, n\}$, when $e(k) \neq 0$, consider two states $s^k$ and $\tilde{s}^k$ such that $s^k D^k_{\underline{\Delta}} \tilde{s}^k$. This implies

that $s^k_{p+1} \leq \tilde{s}^k_{p+1}$ if $k < n$. Moreover, since $s^k \underline{\Delta} \tilde{s}^k$, we have $\log s^k_i \geq \log \tilde{s}^k_i$ $(i = 1, \dots, p)$. Thus, since $\frac{1}{e(k)\log(1+\varepsilon)} > 0$, we obtain $B_i(s^k, e(k)) \geq B_i(\tilde{s}^k, e(k))$ $(i = 1, \dots, p)$. Hence, we get for any $k \in \{1, \dots, n\}$ $s^k D^k_{\underline{\Delta}_\varepsilon} \tilde{s}^k$. □

As a consequence of $(c)$ we have $\underline{\Delta} \subseteq D^n_{\underline{\Delta}_\varepsilon}$ and thus Algorithm 1 using a family of dominance relations $D^k_{\underline{\Delta}_\varepsilon}$, $k = 1, \dots, n$, computes a reduced $(1+\varepsilon)$-approximation (see Theorem 1).

Relation $D^k_{\underline{\Delta}_\varepsilon}$ 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.

Observe that, even if Erlebach et al. (2002) do not explicitly mention the use of a family of dominance relations for $\underline{\Delta}_\varepsilon$, their approach could be restated within Algorithm 1 by using the following family of relations $D^k_E$ defined on $S^k$ by:

$$\text{for all } s^k, \tilde{s}^k \in S^k,\ s^k D^k_E \tilde{s}^k \Leftrightarrow \begin{cases} B_i(s^k, 1/n) = B_i(\tilde{s}^k, 1/n) \quad i = 1, \dots, p \quad \text{and} \\ s^k_{p+1} \leq \tilde{s}^k_{p+1} \end{cases}$$

Remark that $D^k_E \subseteq D^k_{\underline{\Delta}_\varepsilon}$ for $e(k) = 1/n$ $(k = 1, \dots, n)$. This relation, which is quite sufficient to establish the existence of an fptas, has two main disadvantages for an efficient implementation. First, it is very poor since it compares only states lying in the same hyper-rectangle. Therefore, even if two states $s^k$, $\tilde{s}^k$ are such that $s^k D^k_{\underline{\Delta}} \tilde{s}^k$, we keep both of them in $C^k$ provided that they are not in the same hyper-rectangle. Secondly, by applying a constant error of $(1+\varepsilon)^{1/n}$ at each phase, the total error of $1+\varepsilon$ is shared out equitably among all the phases. During the first phases, since the values of the states are small, the hyper-rectangles to which the states belong usually have a length smaller than 1 on all dimensions. In this case, the advantage of the partition is canceled out since only states with same values could be in relation $D^k_E$. Thus, the error allocated to these phases is wasted.

### 4.1.2 Complexity of our approach using $D^k_{\underline{\Delta}_\varepsilon}$

For a given $\varepsilon > 0$, the running time of Algorithm 1 using relation $D^k_{\underline{\Delta}_\varepsilon}$ depends crucially on the error function $e$. In order to guarantee that Algorithm 1 is polynomial both in the size of the instance and in $1/\varepsilon$, we have to add some conditions on the error function aiming at limiting the number of phases with an error equal to 0.

**Definition 5 (Polynomial error function)** *The error function $e$ is a polynomial error function if, for $k = 1, \dots, n$, $e(k) = 1/g(k)$ if $k$ is a multiple of $t$, 0 otherwise, where $t$ is a strictly positive integer in $O(\log n)$ and where, for any $k = 1, \dots, n$, $0 < g(k) \leq cn^d$ for some positive fixed constants $c, d$.*

The following theorem establishes the complexity of Algorithm 1 using the family of dominance relations $D^k_{\underline{\Delta}_\varepsilon}$.

**Theorem 2** *For any $\varepsilon > 0$ and any polynomial error function $e$, Algorithm 1, using the family of dominance relations $D^k_{\underline{\Delta}_\varepsilon}$, is polynomial both in the size of the instance and in $1/\varepsilon$.*

*Proof:* The complexity of Algorithm 1 is in $O(\sum_{k=0}^{n-1}|C^k|^2)$. We study now the cardinality of $C^k$ for $k = 0, \dots, n-1$.

When $e(k) \neq 0$, *i.e.* when $k$ is a multiple of $t$, the cardinality of $C^k$ can be bounded by $\tau(k) = \left(\frac{\log U_{\max}}{e(k)\varepsilon}\right)^p$ with $U_{\max} = \max\{U_1, \dots, U_p\}$ where $U_i$ is an upper bound on the value of the feasible solutions on the $i$th criterion $(i = 1, \dots, p)$.

When $e(k) = 0$, *i.e.* when $k$ is not a multiple of $t$, the cardinality of $C^k$ is at most $2^{k-\ell}$ times the cardinality of $C^\ell$ where $\ell = \lfloor k/t \rfloor t$ is the index of the last phase with an error function different from 0. Hence, in this case, $|C^k|$ can be bounded by $2^{k-\lfloor k/t\rfloor t}\tau(\lfloor k/t \rfloor t)$, where $\tau(0) = |C^0| = 1$.

Finally, the complexity of Algorithm 1 can be bounded by:

$$\sum_{j=0}^{\lfloor n/t \rfloor}\sum_{i=0}^{t-1}\left(2^i\tau(jt)\right)^2 = \sum_{i=0}^{t-1}2^{2i}\sum_{j=0}^{\lfloor n/t \rfloor}\tau(jt)^2 \tag{3}$$

Observe that $\sum_{i=0}^{t-1}2^{2i} = 2^2\sum_{i=0}^{t-1}2^i = 2^2(2^t-1) = 2^{t+2}-4$. Hence, since $t$ is in $O(\log n)$, we get $\sum_{i=0}^{t-1}2^{2i} \leq n^g$ for some fixed positive constant $g$. Thus the complexity of Algorithm 1, that is bounded by (3), is in $O\left(n^g\sum_{j=0}^{\lfloor n/t \rfloor}\tau(jt)^2\right)$.

Since $e$ is a polynomial error function, we have $0 < 1/e(jt) \leq cn^d$ for some positive fixed constants $c$ and $d$, for any $j = 0, \dots, \lfloor n/t \rfloor$. Thus, the complexity of Algorithm 1 is in:

$$O\left(n^{g+2pd+1}\left(\frac{\log U_{\max}}{\varepsilon}\right)^{2p}\right)$$

for some positive fixed constants $d$ and $g$. This establishes that the complexity of Algorithm 1 is polynomial both in the size of the instance and in $1/\varepsilon$. $\square$

Hence, by Theorems 1 and 2 we have that, for any $\varepsilon > 0$ and any polynomial error function $e$, Algorithm 1 using the family of dominance relations $D^k_{\underline{\Delta}_\varepsilon}$ is an fptas that produces a reduced $(1+\varepsilon)$-approximation.

### 4.1.3 Error functions

A crucial parameter of relation $D^k_{\underline{\Delta}_\varepsilon}$ is the error function $e$. The distribution of the error throughout the phases depends on the shape of the error function and on the frequency of phases with a non zero error (see paragraph Shape and frequency). Moreover, the error function considered can be modified during the phases to take into account some particularities of the instance under resolution (see paragraph Strategy of management).

**Shape and frequency** We investigate the following polynomial error functions: for $k = 1, \ldots, n$

- $e_1(k) = 1/(\lfloor n/t \rfloor)$, if $k$ is a multiple of $t$, 0, otherwise
- $e_2(k) = \frac{2k/t}{\lfloor n/t \rfloor (\lfloor n/t \rfloor + 1)}$, if $k$ is a multiple of $t$, 0, otherwise
- $e_3(k) = \frac{6(k/t)^2}{\lfloor n/t \rfloor (\lfloor n/t \rfloor + 1)(2\lfloor n/t \rfloor + 1)}$, if $k$ is a multiple of $t$, 0, otherwise

where $e_1$, $e_2$, and $e_3$ are respectively constant, linear, and quadratic error functions. In the definition of $e_1$, $e_2$, and $e_3$, parameter $t$, which is a strictly positive integer in $O(\log n)$, expresses the frequency of the application of the error. The idea of the frequency is to have some phases with no error in order to apply larger errors to other phases. Of course, in order to remain polynomial in the size of the instance and in $1/\varepsilon$, the number of phases with no error must be limited (see Theorem 2).

In the computational experiments, in Section 5, we show the impact of the error function and of the frequency in our approach.

**Strategy of management** During a phase $k$ a state $\tilde{s}^k \in T^k$ is discarded because of the existence of a state $s^k \in C^k$ such that $s^k D^k_{\underline{\Delta}_\varepsilon} \tilde{s}^k$. Nevertheless, since $D^k_{\underline{\Delta}} \subseteq D^k_{\underline{\Delta}_\varepsilon}$, it could happen that we also have $s^k D^k_{\underline{\Delta}} \tilde{s}^k$. If this happens for all discarded states of $T^k$, it means that the error $e(k)$ attributed to this phase is not used. It is then desirable to redistribute the error $e(k)$ among the remaining phases $k+1, \ldots, n$ for which the error is strictly positive.

This strategy of management of the error is particularly effective during the first phases where states usually have small values and $D^k_{\underline{\Delta}_\varepsilon}$ does not remove more states than $D^k_{\underline{\Delta}}$. All experiments in this paper use this strategy of management of the error.

## 4.2 Complementary dominance relations with respect to $\underline{\Delta}$

Since 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 two other complementary dominance relations for $\underline{\Delta}$. The first one, $D^k_r$, is very easy to establish and the second one, $D^k_b$, although more difficult to establish, is considered owing to its complementarity with $D^k_r$ and $D^k_{\underline{\Delta}_\varepsilon}$.

Dominance relation $D^k_r$ is 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:

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

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.

Dominance relation $D_b^k$ is based on the comparison between extensions of a state and an upper bound of all 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 $u_i \geq s_i^n$, $i = 1, \ldots, p$.

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 $s_i^k + \sum_{j \in J} v_i^j \geq \tilde{u}_i$, $i = 1, \ldots, p$, 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. Let $\mathcal{O}^i$ be an order induced by considering items according to decreasing order of ratios $v_i^k / w^k$ ($i = 1, \ldots, p$). Let $r_i^\ell$ be the rank or position of item $\ell$ in order $\mathcal{O}^i$. Let $\mathcal{O}^{\max}$ be an order according to increasing values of the maximum rank of items in the $p$ orders $\mathcal{O}^i$ ($i = 1, \ldots, p$) where the maximum rank of item $\ell$ 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. Let $\mathcal{O}^{sum}$ be an order according to increasing values of the sum of the ranks of items in the $p$ orders $\mathcal{O}^i$ ($i = 1, \ldots, p$). After relabeling items $k+1, \ldots, n$ according to $\mathcal{O}^{\max}$, completion $J'$ is obtained by inserting sequentially the remaining items into the solution provided that the capacity constraint is respected. $J''$ is defined similarly by relabeling items according to $\mathcal{O}^{sum}$. To compute $u$, we use the classical upper bound presented in (Martello and Toth, 1990, Th. 2.2) computed independently for each criterion value.

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 of Martello and Toth for $\tilde{s}^k$.

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

### 4.3 Use of multiple dominance relations

In order to be efficient, we will use the dominance relations $D_r^k$, $D_{\underline{\Delta}_\varepsilon}^k$, and $D_b^k$ 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 $D_{\underline{\Delta}_\varepsilon}^k$) and then test on a reduced set of states dominance relations requiring a larger computation time (such as $D_b^k$).

## 5 Computational experiments and results

We first present the experimental design (Section 5.1). Then, Section 5.2 is devoted to the presentation of results in the bi-objective case and Section 5.3 to the presentation of results in the tri-objective case. Finally a comparison with an exact method is performed in Section 5.4.

### 5.1 Experimental design

All experiments presented here were performed on a 3.4GHz computer 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 weights, 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 made on instances of type A. Sometimes, other instances such as those of type B, which were introduced in Captivo et al. (2003), are studied. However, instances of type B should be viewed as quasi single-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. In instances of type D, $w^k$ is positively correlated with $v_1^k$ and $v_2^k$. These instances were designed in order to verify if positively correlated weight/values instances are harder than uncorrelated weight/values instances as in the single-criterion context (Martello and Toth, 1990; Kellerer et al., 2004).

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

In the experiments, we also report the results obtained in Bazgan et al. (2009) by using relations $D_r^k$, $D_{\underline{\Delta}}^k$ and $D_b^k$ aiming at solving the exact version of the 0–1 multi-objective knapsack problem. These results are denoted by *exact method*. In this previous work, we showed that the way of ordering items has a dramatic impact on the CPU time; we established experimentally that sorting items according to $O^{\max}$ is much better than using simple orders like $O^{sum}$. Thus, in the following, items are sorted and labeled according to $\mathcal{O}^{\max}$.

Observe finally that all the methods experimented only compute criterion vectors. Standard bookkeeping techniques, not considered here, may be used to produce the corresponding solutions.

### 5.2 Results in the bi-objective case

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

(a) to have a better understanding of the distribution of $ND_\varepsilon$ in the criterion space (see Figure 1)

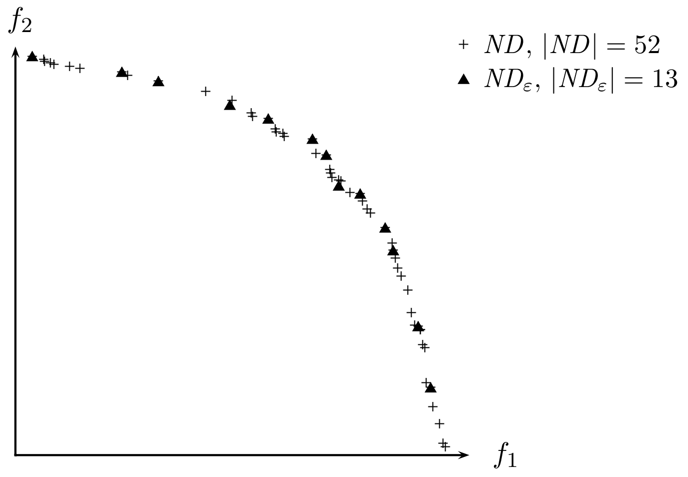
(b) to analyze the impact of the error functions (see Tables 1 and 2)

(c) to evaluate the impact of the variation of $\varepsilon$ in our approach and evaluate the *a posteriori* real error (see Table 3)

(d) to analyze the performance of our approach on large instances (see Table 4)

#### 5.2.1 Distribution of $ND_\varepsilon$ vs $ND$ in the criterion space

In order to appreciate the quality of an $(1 + \varepsilon)$-approximation, we consider a small random instance of type 2A50 and display in Figure 1 a set $ND_\varepsilon$ produced by our approximation algorithm and the set $ND$ obtained by an exact algorithm. We can observe that the size of $ND_\varepsilon$ is much smaller than the cardinality of $ND$ (13 points for $ND_\varepsilon$ vs 52 for $ND$). Moreover, all points of $ND_\varepsilon$ are distributed uniformly along the efficient frontier. Remark also that almost all points of $ND_\varepsilon$ are non-dominated vectors.



Instance 2A50 solved by our approach with $\varepsilon = 0.1$, error function $e_2$, and $t = 1$

Figure 1: Distribution of $ND_\varepsilon$ vs $ND$ in the criterion space

#### 5.2.2 Impact of the error functions

First, we try to determine the best error function to use in relation $D^k_{\underline{\Delta}_\varepsilon}$. In Table 1 we compare the CPU time and the size of $ND_\varepsilon$ obtained for the three polynomial error functions $e_1$, $e_2$, and $e_3$ (see Section 4.1.3). Table 1 shows clearly that the error function has a significant impact on the CPU time and that error function $e_2$ is significantly better for all types of instances. Observe also that error function $e_3$, although less efficient than $e_2$ in terms of CPU time, allows us to generate reduced $(1 + \varepsilon)$-approximations of smaller cardinality. In the following, we will use only error function $e_2$.

Table 1: Impact of different error functions in our approach ($p = 2$)

| type | approximation method | | | | | | exact method | |
|---|---|---|---|---|---|---|---|---|
| | avg. time in s. | | | avg. $\lvert ND_\varepsilon \rvert$ | | | Bazgan et al. (2009) | |
| | $e_1$ | $e_2$ | $e_3$ | $e_1$ | $e_2$ | $e_3$ | avg t. in s. | avg. $\lvert ND \rvert$ |
| 2A-400 | 51.775 | 34.347 | 50.022 | 332.1 | 199.8 | 134.3 | 307.093 | 4631.8 |
| 2B-1000 | 0.238 | 0.180 | 0.299 | 1.0 | 1.0 | 1.0 | 8.812 | 157.0 |
| 2C-300 | 74.308 | 50.265 | 68.974 | 615.3 | 326.4 | 227.5 | 373.097 | 1130.7 |
| 2D-150 | 65.144 | 47.398 | 67.758 | 703.0 | 384.3 | 263.1 | 265.058 | 3418.5 |

$\varepsilon = 0.1$, different error functions, and frequency $t = 1$

Second, we show the impact of the frequency $t$ in the error function $e_2$. Table 2 establishes that our approach is always faster by setting the frequency $t = \lfloor \log n \rfloor$. Observe that the cardinality of $ND_\varepsilon$ is inversely proportional to the frequency $t$. For example the increase of a factor 3 of the frequency (from $t = \lfloor \log n \rfloor$ to $\lfloor 3 \log n \rfloor$) leads to a decrease of about a factor 3 of the size of $ND_\varepsilon$.

Table 2: Impact of the frequency in the error function $e_2$ ($p = 2$)

| | approximation method | | | | | | | | exact method | |
|---|---|---|---|---|---|---|---|---|---|---|
| | avg. time in s. | | | | avg. $\|ND_\varepsilon\|$ | | | | Bazgan et al. (2009) | |
| type | $t = 1$ | $\lfloor \log n \rfloor$ | $\lfloor 2 \log n \rfloor$ | $\lfloor 3 \log n \rfloor$ | $t = 1$ | $\lfloor \log n \rfloor$ | $\lfloor 2 \log n \rfloor$ | $\lfloor 3 \log n \rfloor$ | avg. t. in s. | avg. $\|ND\|$ |
| 2A-400 | 34.347 | 4.536 | 5.441 | 7.664 | 199.8 | 31.3 | 16.1 | 11.9 | 307.093 | 4631.8 |
| 2B-1000 | 0.180 | 0.120 | 0.406 | 1.009 | 1.0 | 1.3 | 1.1 | 1.0 | 8.812 | 157.0 |
| 2C-300 | 50.265 | 7.511 | 8.084 | 11.618 | 326.4 | 53.6 | 27.3 | 18.8 | 373.097 | 1130.7 |
| 2D-150 | 47.398 | 10.874 | 11.935 | 16.156 | 384.3 | 70.1 | 35.8 | 26.4 | 265.058 | 3418.5 |

$\varepsilon = 0.1$, error function $e_2$, and different frequencies

### 5.2.3 Impact of the variation of $\varepsilon$

Table 3: Impact of the variation of the error $\varepsilon$ ($p = 2$)

| type | $n$ | $\varepsilon = 0.1$ | | | $\varepsilon = 0.3$ | | | $\varepsilon = 0.5$ | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | avg. t. | $\|ND_\varepsilon\|$ | avg. error | avg. t. | $\|ND_\varepsilon\|$ | avg. error | avg. t. | $\|ND_\varepsilon\|$ | avg. error |
| A | 100 | 0.042 | 10.1 | 0.0159 | 0.017 | 4.3 | 0.0400 | 0.013 | 2.9 | 0.0638 |
| | 700 | 32.275 | 49.5 | 0.0060 | 10.500 | 18.3 | 0.0200 | 6.605 | 11.2 | 0.0337 |
| B | 1000 | 0.118 | 1.3 | 0.0041 | 0.066 | 1.1 | 0.0102 | 0.052 | 1.0 | 0.0149 |
| | 4000 | 11.220 | 1.8 | 0.0023 | 4.482 | 1.4 | 0.0070 | 3.596 | 1.1 | 0.0119 |
| C | 100 | 0.210 | 25.3 | 0.0178 | 0.077 | 9.4 | 0.0443 | 0.050 | 6.4 | 0.0609 |
| | 500 | 44.368 | 88.3 | 0.0064 | 13.268 | 30.8 | 0.0202 | 7.830 | 19.8 | 0.0336 |
| D | 100 | 2.356 | 52.9 | 0.0183 | 0.675 | 19.5 | 0.0458 | 0.379 | 12.6 | 0.0701 |
| | 250 | 62.970 | 110.9 | 0.0098 | 17.793 | 41.2 | 0.0258 | 9.925 | 25.7 | 0.0449 |

Different error $\varepsilon$, error function $e_2$, and frequency $t = \lfloor \log n \rfloor$
avg. t : average CPU time in second

The results concerning the impact of the variation of $\varepsilon$ in our approach are reported in Table 3. First, observe that the average CPU time is, as expected theoretically, inversely proportional to the value of $\varepsilon$. For example, for instances 2C500 the average CPU time decreases from 44.3s to 7.8s (about a factor 5.5) when increasing the error from $\varepsilon = 0.1$ to 0.5. Second, as also expected theoretically, we observed experimentally that $|ND_\varepsilon|$ is inversely proportional to the value of $\varepsilon$, for all instances except instances of type B (for which $|ND_\varepsilon|$ is already less than 2 when $\varepsilon = 0.1$).

We also give in Table 3, for each series, the "average error" that refers to the *a posteriori* error, which corresponds to the smallest value of $\varepsilon$ such that the returned set is indeed a reduced $(1 + \varepsilon)$-approximation. Observe that the *a posteriori* error is much smaller than the fixed *a priori* error. For example, for instances 2D100, for an *a priori* error $\varepsilon$ set to 0.5, the *a posteriori* error is about 0.07 and tends to decrease with the size of the instances. This clearly shows that, in practice, we can use our approach with large values of $\varepsilon$ in order to obtain very quickly approximations of good quality. The following experiments are performed with an *a priori* error $\varepsilon$ set to 0.1.

### 5.2.4 Performance on large size instances

We present, in Table 4, the performance of our approach on large size instances. The largest instances solved here are those of type B with 20000 items and the instances with the largest reduced $(1+\varepsilon)$-approximations are those of type D with 900 items. Observe that 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 900 items.

Table 4: Results of our approach on large size instances ($p = 2$)

| type | $n$ | time in s. | | | $\lvert ND_\varepsilon \rvert$ | | | avg. |
|---|---|---|---|---|---|---|---|---|
| | | min | avg. | max | min | avg. | max | $\max_k\{\lvert C^k \rvert\}$ |
| A | 100 | 0.024 | 0.042 | 0.072 | 6 | 10.1 | 15 | 2456.7 |
| | 500 | 8.160 | 9.392 | 11.712 | 34 | 39.1 | 46 | 79334.0 |
| | 1000 | 88.121 | 94.050 | 103.402 | 65 | 68.5 | 76 | 321327.6 |
| | 1500 | 355.422 | 369.537 | 414.009 | 87 | 92.0 | 96 | 786580.0 |
| | 2000 | 896.640 | 1030.813 | 1398.060 | 111 | 123.9 | 132 | 1489132.4 |
| | 2500 | 1635.230 | 1917.072 | 2081.410 | 127 | 138.6 | 147 | 2585169.9 |
| B | 1000 | 0.084 | 0.118 | 0.144 | 1 | 1.3 | 2 | 5596.4 |
| | 5000 | 19.814 | 26.062 | 35.422 | 1 | 2.1 | 4 | 252650.7 |
| | 10000 | 245.572 | 269.731 | 318.808 | 2 | 3.3 | 4 | 1160906.4 |
| | 15000 | 654.553 | 896.394 | 1070.930 | 3 | 4.2 | 5 | 2416609.6 |
| | 20000 | 2424.700 | 2816.606 | 3166.580 | 4 | 5.3 | 7 | 5424849.6 |
| C | 100 | 0.140 | 0.210 | 0.316 | 21 | 25.3 | 32 | 9964.2 |
| | 500 | 31.857 | 44.145 | 52.403 | 74 | 88.3 | 104 | 225211.4 |
| | 1000 | 378.135 | 419.595 | 471.269 | 139 | 150.2 | 162 | 923939.4 |
| | 1500 | 1358.300 | 1581.292 | 1801.710 | 194 | 204.3 | 216 | 2205211.0 |
| | 2000 | 3679.770 | 4296.847 | 4749.160 | 255 | 272.0 | 285 | 4256900.6 |
| D | 100 | 1.948 | 2.356 | 2.828 | 50 | 52.9 | 57 | 93507.9 |
| | 300 | 92.433 | 109.082 | 123.271 | 107 | 119.6 | 130 | 1059261.8 |
| | 500 | 605.837 | 640.026 | 681.286 | 185 | 196.4 | 203 | 3034228.2 |
| | 700 | 1861.120 | 1956.371 | 2079.540 | 225 | 241.4 | 251 | 6238134.6 |
| | 900 | 4154.610 | 4689.373 | 5177.200 | 297 | 313.0 | 329 | 10276196.8 |

$\varepsilon = 0.1$, error function $e_2$, and frequency $t = \lfloor \log n \rfloor$

## 5.3 Results in the tri-objective case

In Table 5, we present results of our approach concerning instances of type $A$ and of type $C$ in the tri-objective case. Observe that the size of $ND_\varepsilon$ increases a lot with the addition of a third objective. This explains the variation of the CPU time which is strongly related with the cardinality of $ND_\varepsilon$.

## 5.4 Comparison with an exact method

The results of a comparative study between the exact method presented in Bazgan et al. (2009) and our approximation method using relations $D^k_r$, $D^k_{\underline{\Delta}_\varepsilon}$, and $D^k_b$ are presented in

Table 5: Results of our approach on instances of type $A$ and $C$ in the tri-objective case

| type | $n$ | time in s. | | | $\lvert ND_\varepsilon \rvert$ | | | avg. |
|---|---|---|---|---|---|---|---|---|
| | | min | avg. | max | min | avg. | max | $\max_k\{\lvert C^k \rvert\}$ |
| A | 100 | 0.552 | 5.036 | 10.632 | 36 | 89.0 | 126 | 44660.3 |
| | 150 | 20.221 | 82.051 | 182.235 | 114 | 172.8 | 245 | 183115.4 |
| | 200 | 118.151 | 362.632 | 647.872 | 178 | 267.1 | 404 | 437670.0 |
| | 250 | 582.456 | 1316.778 | 3154.800 | 226 | 320.4 | 407 | 721599.1 |
| C | 10 | <1ms | <1ms | <1ms | 2 | 12.7 | 21 | 51.7 |
| | 50 | 0.300 | 2.424 | 7.688 | 73 | 166.9 | 258 | 25324.5 |
| | 100 | 59.239 | 273.363 | 570.383 | 298 | 450.3 | 621 | 254636.0 |
| | 140 | 1055.520 | 3645.854 | 11071.900 | 552 | 688.0 | 847 | 777715.3 |

$\varepsilon = 0.1$, error function $e_2$ and frequency $t = \lfloor \log n \rfloor$

Tables 6 and 7. We selected this method since, as shown in this previous paper, it is the most efficient exact method currently known. Moreover, the comparison is all the more significant than this exact method can be seen as a degenerate version of our approach where $\varepsilon$ is set to 0.

The two methods have been compared on the same instances and the same computer. Table 6 presents results in the bi-objective case for instances of type A, B, C, and D for increasing size of $n$ for instances that can be solved by the exact method. Table 7 presents results in the tri-objective case for instances of type A for increasing size of $n$ for instances that can be solved by the exact method.

Considering the CPU time, the approximation method is, of course, always faster than the exact method (up to more than 600 times faster in the bi-objective case for instances 2B4000 and up to 356 times in the tri-objective case for instances 3A110). The gap between the CPU time needed by the exact method and the CPU time needed by the approximation method increases with the number of objectives. For example, for instances of type A, in the bi-objective case the decrease is up to a factor 169 (instances 2A700) whereas in the tri-objective case the decrease is up to a factor 356 (instances 3A110).

Observe that, although the cardinality of $ND_\varepsilon$ is very small with regard to the cardinality of $ND$, the quality of the reduced $(1 + \varepsilon)$-approximation is very good since for an *a priori* error $\varepsilon = 0.1$, the *a posteriori* error is always less than 0.02 in the bi-objective case, and varies from 0.0419 to 0.0146 in the tri-objective case. Moreover, as already observed in Section 5.2.3, the *a posteriori* error decreases with the size of the instance.

Table 6: Comparison between the exact method presented in Bazgan et al. (2009) and the approximation method in the bi-objective case ($p = 2$)

| type | $n$ | exact method | | approximation method | | | | |
|---|---|---|---|---|---|---|---|---|
| | | avg. t. in s. | avg. $\lvert ND\rvert$ | avg. t. in s. | | avg. $\lvert ND_{\varepsilon}\rvert$ | | avg. error |
| | 100 | 0.328 | 159.3 | 0.042 | ( $\div$ 8 ) | 10.1 | ( $\div$ 16 ) | 0.0159 |
| A | 400 | 307.093 | 1713.3 | 6.084 | ( $\div$ 50 ) | 31.3 | ( $\div$ 55 ) | 0.0076 |
| | 700 | 5447.921 | 4814.8 | 32.275 | ( $\div$ 169 ) | 49.5 | ( $\div$ 97 ) | 0.0060 |
| | 1000 | 8.812 | 157.0 | 0.118 | ( $\div$ 75 ) | 1.3 | ( $\div$ 121 ) | 0.0041 |
| B | 2000 | 251.056 | 477.7 | 1.452 | ( $\div$ 173 ) | 1.7 | ( $\div$ 281 ) | 0.0028 |
| | 4000 | 6773.264 | 1542.3 | 11.220 | ( $\div$ 604 ) | 1.8 | ( $\div$ 857 ) | 0.0023 |
| | 100 | 2.869 | 558.2 | 0.210 | ( $\div$ 14 ) | 25.3 | ( $\div$ 22 ) | 0.0178 |
| C | 300 | 373.097 | 2893.6 | 10.099 | ( $\div$ 37 ) | 53.6 | ( $\div$ 54 ) | 0.0096 |
| | 500 | 4547.978 | 7112.1 | 44.368 | ( $\div$ 103 ) | 88.3 | ( $\div$ 81 ) | 0.0064 |
| | 100 | 40.866 | 1765.4 | 2.356 | ( $\div$ 17 ) | 52.9 | ( $\div$ 33 ) | 0.0183 |
| D | 200 | 1145.922 | 5464.0 | 36.226 | ( $\div$ 32 ) | 89.4 | ( $\div$ 61 ) | 0.0117 |
| | 250 | 3383.545 | 8154.7 | 62.970 | ( $\div$ 54 ) | 110.9 | ( $\div$ 74 ) | 0.0098 |

Approximation: $\varepsilon = 0.1$, error function $e_2$, and frequency $t = \lfloor \log n \rfloor$
The decrease factors of the avg. CPU time and of the size of the returned set, corresponding respectively to avg. t. in s. of exact method / avg. t. in s. of approximation method and $|ND|/|ND_{\varepsilon}|$, are given in brackets

Table 7: Comparison between the exact method presented in Bazgan et al. (2009) and the approximation method in the tri-objective case ($p = 3$)

| type | $n$ | exact method | | approximation method | | | | |
|---|---|---|---|---|---|---|---|---|
| | | avg. t. in s. | avg. $\lvert ND\rvert$ | avg. t. in s. | | avg. $\lvert ND_{\varepsilon}\rvert$ | | avg. error |
| | 10 | <1ms | 8.3 | <1ms | – | 4.8 | ( $\div$ 2 ) | 0.0419 |
| | 30 | 0.012 | 112.9 | 0.006 | ( $\div$ 2 ) | 22.2 | ( $\div$ 5 ) | 0.0246 |
| A | 50 | 0.611 | 540.6 | 0.077 | ( $\div$ 8 ) | 39.5 | ( $\div$ 14 ) | 0.0207 |
| | 70 | 16.837 | 1384.4 | 0.451 | ( $\div$ 37 ) | 45.9 | ( $\div$ 30 ) | 0.0210 |
| | 90 | 538.768 | 4020.3 | 3.558 | ( $\div$ 151 ) | 92.3 | ( $\div$ 44 ) | 0.0160 |
| | 110 | 3326.587 | 6398.3 | 9.347 | ( $\div$ 356 ) | 108.8 | ( $\div$ 59 ) | 0.0146 |
| | 10 | <1ms | 17.7 | <1ms | – | 12.7 | ( $\div$ 1 ) | 0.0323 |
| | 20 | 0.030 | 300.2 | 0.014 | ( $\div$ 2 ) | 70.1 | ( $\div$ 4 ) | 0.0362 |
| C | 30 | 0.431 | 649.1 | 0.093 | ( $\div$ 5 ) | 95.3 | ( $\div$ 7 ) | 0.0325 |
| | 40 | 3.684 | 1538.9 | 0.346 | ( $\div$ 11 ) | 113.1 | ( $\div$ 14 ) | 0.0260 |
| | 50 | 83.594 | 3650.9 | 2.424 | ( $\div$ 34 ) | 166.9 | ( $\div$ 22 ) | 0.0262 |
| | 60 | 2572.981 | 9647.9 | 15.027 | ( $\div$ 171 ) | 287.3 | ( $\div$ 34 ) | 0.0207 |

Approximation: $\varepsilon = 0.1$, error function $e_2$, and frequency $t = \lfloor \log n \rfloor$
The decrease factors of the avg. CPU time and of the size of the returned set, corresponding respectively to avg. t. in s. of exact method / avg. t. in s. of approximation method and $|ND|/|ND_{\varepsilon}|$, are given in brackets

### 5.4.1 Comparison with the approximation method of Erlebach et al. (2002)

The results of a comparative study, in the bi-objective case, between the approximation method of Erlebach et al. (2002) and our approximation method are presented in Table 8. We also give in this table the results for the exact method presented in Bazgan et al. (2009).

The approximation method of Erlebach et al. (2002) can be restated in Algorithm 1 by using the family of dominance relations $D_E^k$ defined in Section 4.1. Observe that, since $\underline{\Delta} \not\subseteq D_E^n$, the second part of Theorem 1 does not hold. Thus the approximation method of Erlebach et al. (2002) provides us with an $(1+\varepsilon)$-approximation but not a reduced $(1+\varepsilon)$-approximation. This could be corrected easily by filtering the approximation in order to eliminate dominated solutions.

Table 8: Comparison between the approximation method of Erlebach et al. (2002) and our approximation method in the bi-objective case ($p = 2$)

| type | $n$ | Erlebach et al. (2002) | | Our approximation method | | | exact method | | |
|---|---|---|---|---|---|---|---|---|---|
| | | avg. t. in s. | avg. $\|output\|$ | avg. t. in s. | | avg. $\|ND_\varepsilon\|$ | avg. t. in s. | | avg. $\|ND\|$ |
| A | 10 | <1ms | 503.6 | <1ms | – | 2.3 | <1ms | – | 3.6 |
| | 50 | 15.825 | 1402686.9 | 0.006 | ( ÷ 2826 ) | 7.2 | 0.016 | ( ÷ 965 ) | 43.4 |
| | 110 | 536.577 | 13161801.6 | 0.076 | ( ÷ 7023 ) | 11.6 | 0.628 | ( ÷ 854 ) | 167.6 |
| B | 10 | <1ms | 444.4 | <1ms | – | 1.0 | <1ms | – | 1.2 |
| | 50 | 2.616 | 223353.8 | 0.001 | ( ÷ 2616 ) | 1.0 | 0.001 | ( ÷ 3270 ) | 3.0 |
| | 110 | 70.466 | 1971772.1 | 0.002 | ( ÷ 35233 ) | 1.0 | 0.002 | ( ÷ 29361 ) | 6.3 |
| C | 10 | <1ms | 482.8 | <1ms | – | 6.0 | <1ms | – | 12.5 |
| | 50 | 19.261 | 1533844.2 | 0.024 | ( ÷ 803 ) | 15.3 | 0.084 | ( ÷ 228 ) | 176.4 |
| | 110 | 653.325 | 14033379.6 | 0.242 | ( ÷ 2704 ) | 25.1 | 3.493 | ( ÷ 187 ) | 578.9 |
| D | 10 | <1ms | 484.2 | <1ms | – | 10.7 | <1ms | – | 31.6 |
| | 50 | 17.549 | 1454469.5 | 0.198 | ( ÷ 89 ) | 35.7 | 1.372 | ( ÷ 13 ) | 581.6 |
| | 110 | 566.662 | 13333536.4 | 3.613 | ( ÷ 157 ) | 58.7 | 61.618 | ( ÷ 9 ) | 1963.3 |

Our approximation method: $\varepsilon = 0.1$, error function $e_2$, and frequency $t = \lfloor \log n \rfloor$
The decrease factors of the avg. CPU time, corresponding to avg. t. in s. of Erlebach et al. (2002) / avg. t. in s. of our approximation method and of the exact method, are given in brackets

The three methods have been compared on the same instances and the same computer. We implemented the approximation method of Erlebach et al. (2002) in C++. Nevertheless, we used an AVL tree to store states at each phase, since it is impossible, for memory reasons, to store the states in an array, as suggested by the authors. This leads to increase the running time to $O(n^p(n \log U_{\max}/\varepsilon)^p \log(n \log U_{\max}/\varepsilon))$.

Considering the CPU time, we can conclude that our approach is always extremely faster than the method of Erlebach et al. (2002) on the considered instances. More interestingly, even our exact method performs faster than the approximation method of Erlebach et al. (2002). Observe, however, that the approximation method of Erlebach et al. (2002) is less sensitive to the type of instances than our method since it performs quite similarly on the instances of types A, C, or D.

## 6 Conclusions

The purpose of this work was to design a practically efficient fptas, based on a dynamic programming algorithm, for solving the approximation version of the 0–1 multi-objective knapsack problem. We showed indeed that by using several complementary dominance relations, and sharing the error appropriately among the phases, we obtain an fptas which is experimentally extremely efficient. The practical use of this approach depends on the requirements imposed by the user. If the user wants to be sure about the quality of the approximation, he/she should set low values for $\varepsilon$, *e.g.* $\varepsilon = 0.1$, in order to obtain reasonably fast an excellent approximation. Alternatively, he/she could overestimate the value of $\varepsilon$, *e.g.* by setting $\varepsilon = 0.5$, in order to obtain extremely fast a reasonably good approximation. In the latter case, our approach becomes competitive with metaheuristics, with the additional advantage of a theoretical *a priori* guarantee and a much better practical *a posteriori* error.

While we focused in this paper on the approximation version of the 0–1 multi-objective knapsack problem, we could envisage in future research to apply dominance relations based on similar ideas to the approximation version of other multi-objective problems which admit dynamic programming formulations, such as the multi-objective shortest path problem or multi-objective scheduling problems.

## References

Angel, E., Bampis, E., and Kononov, A. (2003). On the approximate tradeoff for bicriteria batching and parallel machine scheduling problems. *Theoretical Computer Science*, 306(1-3):319–338.

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

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

Ehrgott, M. (2005). *Multicriteria optimization.* Springer, Berlin.

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

Kellerer, H., Pferschy, U., and Pisinger, D. (2004). *Knapsack Problems.* Springer, Berlin.

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

Martello, S. and Toth, P. (1990). *Knapsack Problems.* Wiley, New York.

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

Papadimitriou, C. H. and Yannakakis, M. (2000). On the approximability of trade-offs and optimal access of web sources. In *IEEE Symposium on Foundations of Computer Science (FOCS 2000), Redondo Beach, California*, pages 86–92.

Safer, H. M. and Orlin, J. B. (1995a). Fast approximation schemes for multi-criteria combinatorial optimization. Working Paper 3756-95, Sloan School of Management, Massachusetts Institute of Technology.

Safer, H. M. and Orlin, J. B. (1995b). Fast approximation schemes for multi-criteria flow, knapsack, and scheduling problems. Working Paper 3757-95, Sloan School of Management, Massachusetts Institute of Technology.

Serafini, P. (1986). Some considerations about computational complexity for multiobjective combinatorial problems. In Jahn, J. and Krabs, W., editors, *Recent advances and historical development of vector optimization*, volume 294 of *LNEMS*, pages 222–232, Berlin. Springer-Verlag.

Warburton, A. (1987). Approximation of Pareto optima in multiple-objective, shortest-path problems. *Operations Research*, 35(1):70–79.

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