Algorithmic Improvements on Dynamic Programming for the Bi-objective {0,1} Knapsack Problem

José Rui Figueira[1], Luís Paquete[2], Marco Simões[2], and Daniel Vanderpooten[3]

[1]INPL, Ecole des Mines de Nancy, Laboratoire LORIA, France
(Ass. member at CEG-IST, Instituto Superior Técnico, Lisbon)
Jose.Figueira@mines.inpl-nancy.fr

[2]CISUC, Department of Informatics Engineering,
University of Coimbra, Portugal.
paquete@dei.uc.pt, msimoes@student.dei.uc.pt

[3]University Paris-Dauphine, LAMSADE, Paris, France
vdp@lamsade.dauphine.fr

Abstract

This paper presents several methodological and algorithmic improvements over a state-of-the-art dynamic programming algorithm for solving the bi-objective $\{0,1\}$ knapsack problem. The variants proposed make use of new definitions of lower and upper bounds, which allow a large number of states to be discarded. The computation of these bounds are based on the application of dichotomic search, definition of new bound sets, and bi-objective simplex algorithms to solve the relaxed problem. Although these new techniques are not of a common application for dynamic programming, we show that the best variants tested in this work can lead to an average improvement of 20% in CPU-time and significant less memory usage than the original approach in a wide benchmark set of instances, even for the most difficult ones in the literature.

1 Introduction

The single-objective $\{0,1\}$ knapsack problem consists of choosing a subset of objects from a finite set that maximizes the overall profit, which results from the sum of the individual benefits of the selected objects where a capacity constraint must be fulfilled, i.e., the sum of the weights of the selected objects must not surpass a given capacity [12, 14].

The multi-objective version of this problem consists of "maximizing" a finite small number of objective functions under the same budgetary constraint. The operator "maximizing" means to search for all efficient (nondominated) solution (vectors). The expression "efficient solutions" is related to the space of objects while the expression "nondominated vectors" is related to the space of the outcomes or objective functions values. A vector is said to be *nondominated* if there is no other vector that improves simultaneously the outcomes of all the objectives.

The problem of searching for efficient solutions can be stated as a $\{0,1\}$ linear programming problem as follows:

$$\begin{array}{rl} \text{“max”} & f(x) = \left(f^1(x), f^2(x), \ldots, f^\ell(x), \ldots, f^p(x)\right) \\ \text{subject to:} & w^{\mathrm{T}} x \leqslant W, x \in \{0,1\}^n \end{array} \quad (1)$$

where $f^\ell(x) = \sum_{j=1}^{n} v_j^\ell x_j$, v_j^ℓ is the benefit of object j on objective ℓ, $j = 1, \ldots, n$, $\ell = 1, \ldots, p$, and $x = (x_1, \ldots, x_j, \ldots, x_n)$ with $x_j = 1$ if object j is included in the subset of the selected objects and $x_j = 0$ otherwise; $w = (w_1, \ldots, w_j, \ldots, w_n)$ is the weight vector and W is the overall capacity. Operator "max" means that it is not possible to maximize all the objectives simultaneously. Assume, without loss of generality, that v_j^ℓ, w_j, and W belong to $\mathbb{N}$, $j = 1, \ldots, n$, $\ell = 1, \ldots, p$ and that $\sum_{j=1}^{n} w_j > W$ (to avoid obvious solutions). In this paper, we consider the bi-objective $\{0, 1\}$ knapsack problem, i.e., the multi-objective $\{0, 1\}$ knapsack problem with $p = 2$. Applications of this problem can be found in capital budgeting [17, 10], transportation [19], and biology [13].
Several approaches were developed to deal with the multi-objective $\{0, 1\}$ knapsack problem. Some are of an exact nature, such as branch-and-bound algorithms [20] and dynamic programming (DP) [11, 4, 2], while others are approximate approaches including polynomial approximations [6, 3], heuristics [7], metaheuristics [8], and hybrid methods [9]. Exact based algorithms, such as DP, can deal with different and difficult medium-large size instances in a very efficient way [2], while approximation based algorithms appear to be more relevant for very large scale problems.
This article deals with several improvements of DP algorithm of Bazgan et al. [2] for the bi-objective $\{0, 1\}$ knapsack problem (herewith called BKP-DP). This approach uses several complementary dominance relations to discard elements from the current pool of partial solutions at each stage. For one of the three dominance relations, the algorithm needs to update two bounds at each stage: an *upper bound* vector obtained for each partial solution in the pool based on the improved state-of-the-art Martello and Toth bound [14], and a *lower bound set* that contains feasible extensions of partial solutions in the pool. A partial solution is discarded if its upper bound is dominated by some extension in a given lower bound set. Clearly, the tighter the lower and the upper bound are, the larger the number of partial solutions that can be discarded. In this article, we present three ways of tightening the upper bound and the lower bound set. Briefly, the three techniques are described as follows:

- Variant 1, which computes of a small set of efficient solutions by solving a sequence of weighted scalarized bi-objective $\{0, 1\}$ knapsack problem with dichotomic search.

- Variant 2, which computes the nondominated extreme vectors of the relaxation of the problem solved by a bi-objective simplex algorithm and makes use of a repairing mechanism (the *improvement method* described by Gomes da Silva et al. [8]) to restore feasibility.

- Variant 3, which generates feasible extensions for each partial solution at a given stage k by adding its profit to the profits obtained through dichotomic search on the reduced bi-objective $\{0, 1\}$ knapsack problem with the remaining $n - k$ objects.

Note that Variants 1 and 2 will generate a lower bound, whereas Variant 3 will generate an upper bound for each partial solution. In addition, note that Variant 1 and 3 consist of solving several single-objective $\{0, 1\}$ knapsack problems. However, Variant 1 performs dichotomic search once and Variant 3 is only performed in the last stages.
In this article, we incorporate the computation of these bounds in the state-of-the-art DP algorithm proposed by Bazgan et al. [2] and analyze, from an experimental point of view, the performance of several variants on a wide benchmark set of instances. The paper is organized as follows. Section 2 introduces the main definitions and notation in multi-objective optimization as well as an explanation of BKP-DP algorithm. The three variants used for improving the BKP-DP algorithm are presented in Section 3. Section 4 comprises the design of the experiments, computational results and discussion. Finally, Section 5 provides conclusions and lines for future research.

2 Main concepts, definitions, and notation

This section is devoted to the main concepts in multi-objective combinatorial optimization, their definitions, notation as well as a brief description of BKP-DP algorithm proposed by Bazgan et al. [2].

2.1 Optimality concepts

Let X denote the set of feasible solutions in the object or decision space and Y their image, $Y := f(X)$, in the objective space. Let $x, x' \in X$. We introduce the following dominance relations for p objective functions:

- $f(x) \underline{\Delta} f(x')$ (x dominates x'), if and only if $f_i(x) \geqslant f_i(x')$, $i = 1, \ldots, p$;
- $f(x) \Delta f(x')$ (x strictly dominates x'), if and only if $x \underline{\Delta} x'$ and $f(x) \neq f(x')$.

A solution $x \in X$ is *efficient* if and only if there is no other feasible solution $x' \in X$ such that $f(x') \Delta f(x)$ and its corresponding objective vector is *nondominated*. The set of all efficient solutions is called the *efficient set* and the set of all nondominated vectors is denoted by $ND(Y)$. The algorithms described in this article output the set $ND(Y)$. The computation of the efficient set can be obtained by keeping states with the same profit and weight.
Very often in multi-objective combinatorial optimization it is usual to distinguish between supported and unsupported vectors (solutions) in the objective (decision) space [18]. Consider $Conv(Y)$ the convex hull of Y, $Bound(Conv(Y))$ the boundary of $Conv(Y)$, and $Int(Conv(Y))$ the interior of $Conv(Y)$. A nondominated supported vector, $f(x)$, is a nondominated vector located in $Bound(Conv(Y))$, while an unsupported nondominated vector $f(x')$ belongs to $Int(Conv(Y))$. The preimage of supported (unsupported) vectors are called supported (unsupported) efficient solutions. When necessary, the notation $NDS(Y)$ and $NDU(Y)$ is used to denote the set of all supported nondominated vectors and all unsupported nondominated vectors, respectively.

2.2 Main features of BKP-DP algorithm

The sequential process used in BKP-DP algorithm consists of n stages. At any stage k, the algorithm generates a set of states S_k that correspond to a subset of the feasible solutions made up of objects belonging exclusively to the k first objects, $k = 1, \ldots, n$. A state $s = \left(s^1, s^2, s^3\right) \in S_k$ represents a feasible solution of profits s^1 and s^2, and weight s^3. At each stage $k = 1, \ldots, n$, the states are generated according to the following recursion:

$$S_k := Dom\left(T_k := S_{k-1} \cup \left\{\left(s^1 + v_k^1, s^2 + v_k^2, s^3 + w_k\right) : s^3 + w_k \leqslant W, s \in S_{k-1}\right\}\right) \tag{2}$$

with the basis case $S_0 := \{(0, 0, 0)\}$. $Dom(T_k)$ denotes the use of certain dominance relations that allow to *filter* the states in T_k. The efficiency of BKP-DP depends strongly on the definition of these relations. Bazgan et al. propose the use of three dominance relations at each stage $k = 1, \ldots, n$ that allow to discard states in T_k that will not lead to other states that represent nondominated vectors. At stage $k = 1, \ldots, n$, considering two states $s \in T_k$ and $\bar{s} \in T_k$, s is discarded, and thus does not belong to S_k, if and only if at least one of the following three conditions holds:

- **(D1)** $\bar{s} = \left(s^1 + v_k^1, s^2 + v_k^2, s^3 + w_k\right)$ and $s^3 \leqslant W - \sum_{j=k}^{n} w_j$,
- **(D2)** $\left(\bar{s}^1, \bar{s}^2\right) \underline{\Delta} \left(s^1, s^2\right)$ and $\bar{s}^3 \leqslant s^3$ if $k < n$,

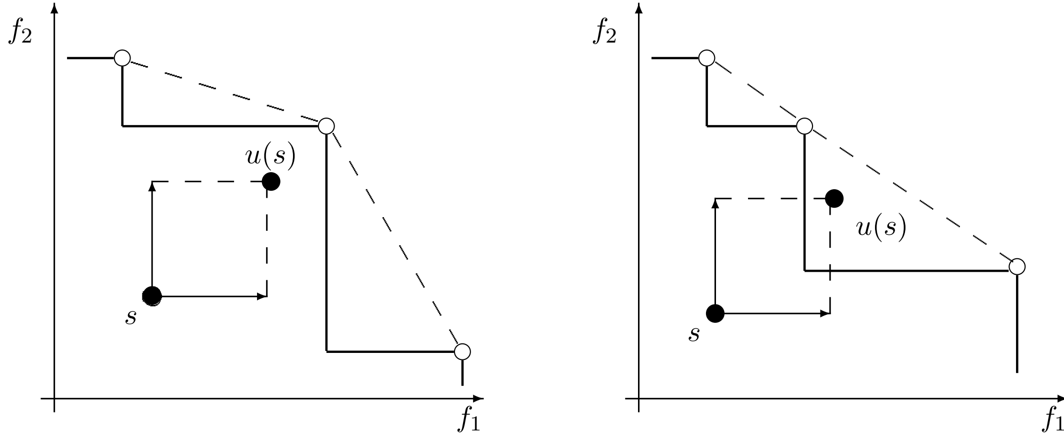

Figure 1: Illustration of condition (D3).

- **(D3)** $\left(\bar{s}^1 + \sum_{j \in J} v_j^1,\, \bar{s}^2 + \sum_{j \in J} v_j^2\right) \underline{\Delta}\, u(s)$ and $\bar{s}^3 \leqslant W - \sum_{j \in J} w_j$,

where $u(s)$ is an upper bound of state s and J corresponds to the indices of the $n-k$ last objects with respect to given pre-ordering of objects provided that the capacity constraint is not violated.
Condition (D1) discards states whose residual capacity exceeds or is equal to the sum of the weights of the remaining objects. Condition (D2) generalizes the dominance relation used within the Nemhauser-Ullman algorithm [15]. Condition (D3) discards states based on the comparison between lower and upper bounds (right and left-hand side of the first condition of D3, respectively). At each comparison, a lower bound set is given by considering all states $\bar{s} \in T_k$. Figure 1 provides an illustration of this condition where white circles correspond to the lower bound set. In the left-hand-side plot, state s is discarded since $u(s)$ is dominated by a lower bound vector. In the right-hand-side plot, state s is not discarded since $u(s)$ is not dominated by any lower bound vector.
The upper bound $u(s) := (u^1, u^2)$ in condition (D3) is computed according to the improved Martello and Toth bound [14]. Let $\overline{W}(s) := W - s^3$ be the residual capacity associated to state $s \in S_k$. We denote by c_i the position of the first object in $\{k+1, \ldots, n\}$ that cannot be added to s due to the capacity constraint, when the objects are ordered according to a given criterion. Thus, given the same ordering of the objects, the upper bound u^i, $i = 1, 2$, is computed as follows:

$$u^i := s^i + \sum_{j=k+1}^{c_i - 1} v_j^i + \max\left\{ \left\lfloor \overline{W}(s) \frac{v_{c_i+1}^i}{w_{c_i+1}} \right\rfloor, \left\lfloor v_{c_i}^i - \left(w_{c_i} - \overline{W}(s)\right) \frac{v_{c_i-1}^i}{w_{c_i-1}} \right\rfloor \right\} \tag{3}$$

Two orderings of the objects were considered by the authors. Let O_i corresponds to the sequence of objects ordered according to the ratios v_k/w_k for objective i and let r_i be the rank of a given object with respect to objective i, $i = 1, 2$. O_{sum} denotes an order according to increasing values of the sum of the positions of objects in orders O_1 and O_2. $O_{\max}$ denotes an order according to the increasing values of the maximum rank of objects in orders O_1 and O_2, where the worst rank of an object in the orders above is given by $\max\{r_1, r_2\} + (r_1 + r_2)/2n$.
Bazgan et al. [2] showed that is possible to devise a DP algorithm that operates under the three dominance conditions and orderings as described above and that S_n will contain all and only the elements of $ND(Y)$. For efficiency reasons, the authors suggest to test conditions (D1) and (D2) first since they are easier to check. Condition (D3), which is more computationally expensive, is a final step at each stage of the algorithm.

3 Algorithmic variants

In this study, three variants are proposed that explore three different ways of defining upper and lower bounds and that allow to discard further states. Therefore, they redefine condition (D3) as described in Section 2.2 and include a pre-processing step.

3.1 Variant BKP-DP1

In variant BKP-DP1, a lower bound set is given by the set of supported solutions of the bi-objective $\{0,1\}$ knapsack problem that is computed in a pre-processing step. This is obtained by performing *dichotomic search* [1]. This procedure consists of constructing a weighted sum objective function of the bi-objective $\{0,1\}$ knapsack problem and optimize according to different weights. The following procedure guarantees that set $NDS(Y)$ is found:

1. Compute the lexicographic maximal (lexmax) solutions x_1 and x_2 w.r.t. f^1 and f^2, respectively. Let $x_1 \in \arg\operatorname{lexmax}\{(f^1(x), f^2(x)) : x \in X\}$ and $x_2 \in \arg\operatorname{lexmax}\{(f^2(x), f^1(x)) : x \in X\}$. Let $y_1 := f(x_1)$, $y_2 := f(x_2)$, $V := \emptyset$ and $k := 2$.
2. Let $R := \{y_1, \ldots, y_k\}$ with $y_1^1 > y_2^1 > \ldots > y_k^1$. If $R \setminus V = \{y_k\}$, then stop; otherwise let $y_i \in \arg\max\{y^1 : y \in R \setminus V\}$.
3. Let $\lambda^1 := y_{i+1}^2 - y_i^2$ and $\lambda^2 := y_i^1 - y_{i+1}^1$. Let $v_j^* = \lambda^1 v_j^1 + \lambda^2 v_j^2$, for $j = 1, \ldots, n$.
4. Compute the (single-objective) optimal solution $\bar{x}$ with respect to profits v^*. If $f(\bar{x}) = y_i$ or $f(\bar{x}) = y_{i+1}$, then $V := V \cup y_i$; otherwise, let $y_{k+1} := f(\bar{x})$ and $R := R \cup y_{k+1}$. Let $k = k + 1$ and go to Step 2.

At the end of above procedure, set R will contain only and all elements of $NDS(Y)$.
For this variant, we rewrite condition (D3) as given in Section 2.2 as follows: At a stage $k = 1, \ldots, n$, a state $s \in T_k$ is removed if there exists a vector $\bar{s} \in NDS(Y)$ such that $\bar{s} \underline{\Delta} u(s)$. Note that, as opposed to the original algorithm proposed by Bazgan et al. [2] where a lower bound set is updated for each partial solution in the pool (see condition D3 in Section 2.2), the supported solutions need only to be computed once in a pre-processing step. However, this implies that an NP-hard problem has to be solved for each weighted sum problem. In the implementation used in this study, the code of Pisinger [16] was adapted to solve each weighted sum problem. Preliminary computational results on a wide benchmark set of problems indicated that this implementation is very fast in practice.

3.2 Variant BKP-DP2

Variant BKP-DP2 also computes a lower bound set as variant BKP-DP1, but avoids the drawback of solving an NP-hard problem as described in the previous section. This is performed by solving the relaxation of the bi-objective $\{0,1\}$ knapsack problem with a simplex algorithm and restoring the feasibility of the relaxed solutions. The simplex algorithm described as follows computes all the nondominated extreme vectors as well as efficient solutions of the bi-objective problem. Hence, it is a more complete version of the algorithm presented in Gomes da Silva et al. [8] that was used to compute only some extreme non-dominated vectors.
The continuous $0-1$ bi-objective knapsack problem can be formalized as follows (see Section 1 for the notation and Eq. (1) for the discrete variant):

$$\begin{aligned} \max \quad & f(x) = \big(f^1(x), f^2(x)\big) \\ \text{subject to:} \quad & w^{\mathrm{T}} x \leqslant W, x \in [0,1]^n \end{aligned} \tag{4}$$

The simplex algorithm computes the set of all nondominated vectors of problem (4). It start from one lexicographically-optimal solution. Then, it proceeds with an efficient pivoting process until an efficient solution of optimal value for objective f^2 is obtained. In order to obtain the lexicographically-optimal solution, the objects are sorted in non-increasing order of the profit-to-weight ratio v_k^1/w_k. In case of ties, the objects with the same profit-to-weight ratio are sorted in non-increasing order of the profit-to-weight ratio for v_k^2/w_k.

Remark 3.1 *If two objects j and ℓ have the same profit-to-ratio on the first and the second objective, then consider only one object of weights $w_j + w_\ell$ and of profit $(v_j^1 + v_\ell^1, v_j^2 + v_\ell^2)$.*

The lexicographical-optimal solution x^* is obtained by using a greedy procedure that includes sorted objects until the capacity is achieved. Let $c = \min\left\{j : \sum_{k=1}^{j} w_j > W\right\}$ be the critical object; then $x^* = (x_1^*, \ldots, x_n^*)$ is such that:

$$x_j^* = \begin{cases} 1 & \text{for } j = 1, \ldots, c-1, \\ \left(W - \sum_{k=1}^{c-1} w_k\right)/w_c, & \text{for } j = c, \\ 0 & \text{for } j = c+1, \ldots, n. \end{cases}$$

Given that x_c^* is not equal to zero (that case will be considered later on), it constitutes an efficient basic variable. Note that an efficient basis is constituted by only one efficient basic variable.
The algorithm uses a pivoting procedure that ensures the transition from one efficient solution to an adjacent one that decreases the value on the first objective. This procedure, which starts with solution x^* and one of its associated efficient basis $\{x_c^*\}$, is described as follows for the general case considering an efficient solution x and one of its associated efficient basis $\{x_c\}$.

1. Given that variable x_c is basic, compute the reduced costs associated to variable x_j on the objective i that is $V_{(c,j)}^i = v_j^i - v_c^i \cdot w_j/w_c$, $i = 1, 2$ and $j = 1, \ldots, n$. The indexes of candidates variables to enter the basis are:

$$J = \left\{ j \in \{1, \ldots, n\} \setminus c \text{ such that: } \begin{array}{ll} V_{(c,j)}^1 < 0 \text{ and } V_{(c,j)}^2 > 0, & \text{if } x_j = 0 \\ V_{(c,j)}^1 > 0 \text{ and } V_{(c,j)}^2 < 0, & \text{if } x_j = 1 \end{array} \right\}$$

   If J is an empty set, the algorithm terminates since it is not possible to increase the value of solution x on the second objective. Hence, x is an efficient solution that is optimal for the second objective. Otherwise, the algorithm proceeds with an efficient pivoting between x_c and variable x_{j^*} ($j^* \in J$), which provides a new efficient solutions such that the slope of the line through this new solution and the previous solution x is the smallest. Then, j^* is such that

$$\frac{V_{(c,j^*)}^2}{V_{(c,j^*)}^1} = \min_{j \in J} \frac{V_{(c,j)}^2}{V_{(c,j)}^1}$$

   Remark 3.2 *Since the objects of the same profit-to-weight ratio for all $j \in \{1, \ldots, n\} \setminus c$ were added, it does not hold that $V_{(c,j)}^1 = 0$ and $V_{(c,j)}^2 = 0$.*

   Remark 3.3 *However, it is possible to obtain variables such that $V_{(c,j)}^1 = 0$ and $V_{(c,j)}^2 < 0$ or $V_{(c,j)}^1 < 0$ and $V_{(c,j)}^2 = 0$, for $x_j = 0$, as well as variables such that $V_{(c,j)}^1 = 0$ and $V_{(c,j)}^2 > 0$ or $V_{(c,j)}^1 > 0$ and $V_{(c,j)}^2 = 0$, for $x_j = 1$. Note that these variables are not candidate to enter the basis since they cannot lead to an efficient pivoting and to a new efficient solution.*

   Remark 3.4 *Index j^* is the index in J that maximizes the value of $-V_{(c,j)}^2/\left(V_{(c,j)}^1 - V_{(c,j)}^2\right)$*

2. Let $\delta = w_c/w_j$. The values of variables x_c and x_j are updated as follows.

$$
\text{if } x_j = 0 \begin{cases} \text{if } \delta x_c < 1 & \begin{cases} x_j := \delta x_c \\ x_c := 0 \\ x_j \text{ enters the basis in substitution of } x_c \end{cases} \\ \text{if } \delta x_c > 1 & \begin{cases} x_j := 1 \\ x_c := x_c - \delta \\ x_c \text{ remains in the basis} \end{cases} \\ \text{if } \delta x_c = 1 & \begin{cases} x_j := 1 \\ x_c := 0 \\ \text{The solution is integer} \end{cases} \end{cases}
$$

$$
\text{if } x_j = 1 \begin{cases} \text{if } \delta(1 - x_c) < 1 & \begin{cases} x_j := 1 - \delta(1 - x_c) \\ x_c := 1 \\ x_j \text{ enters the basis in substitution of } x_c \end{cases} \\ \text{if } \delta(1 - x_c) > 1 & \begin{cases} x_j := 0 \\ x_c := x_c + \delta \\ x_c \text{ remains in the basis} \end{cases} \\ \text{if } \delta(1 - x_c) = 1 & \begin{cases} x_j := 0 \\ x_c := 1 \\ \text{The solution is integer} \end{cases} \end{cases}
$$

The final solution is efficient. The pivoting process is repeated with the new basic solution. In case of degeneration, i.e., when we obtain an integer solution, the pivoting procedure is slightly different. We consider all possible increasing pivots between x_i and x_j for $i = 1, \ldots, n$ and $j = i+1, \ldots, n$ satisfying that

$$
\begin{array}{l} V^1_{i,j} > 0 \text{ and } V^2_{i,j} < 0, \text{ if } x_i = 0 \text{ and } x_j = 1 \\ V^1_{i,j} < 0 \text{ and } V^2_{i,j} > 0, \text{ if } x_i = 1 \text{ and } x_j = 0 \end{array} \tag{5}
$$

Among all these possible pivots satisfying (5), the algorithm pivots between x_c and x_{j^*} such that

$$
\frac{V^2_{(c,j^*)}}{V^1_{(c,j^*)}} = \min \frac{V^2_{(c,j)}}{V^1_{(c,j)}}
$$

Hence, the algorithm updates the values of x_c and x_{j^*} using step 2 described above.
The computation of all the extreme nondominated vectors in the objective space defines an upper bound set for $ND(Y)$. From these extreme points, a set of feasible vectors can be obtained and used as a pool of solutions for the dynamic programming approach. The improved method to restore feasibility is the one presented by Gomes da Silva et al. [8], the *improvement method for the scatter search algorithm*. For each solution obtained by the simplex method described above, this repair method iteratively removes the objects with the lowest profit-to-weight ratio to each objective, until feasibility is achieved. Then, at each feasible solution, objects are inserted according to the non-increasing order of the profit-to-weight ratio for each objective. The final set of solutions become a lower bound set for the original bi-objective problem.
Let B denote the set of vectors returned by the procedure above. The condition (D3) as given in Section 2.2 becomes as follows: At a stage $k = 1, \ldots, n$, a state $s \in T_k$ is removed if there exists a

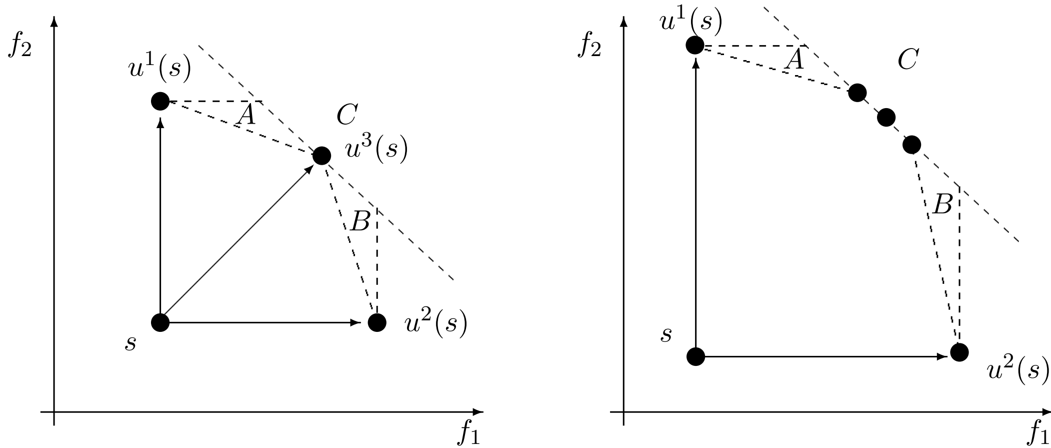

Figure 2: Illustration of upper bounds in BKP-DP3.

vector $\bar{s} \in B$ such that $\bar{s}\underline{\Delta}u(s)$. Similarly to the procedure described in Section 3.1, set B is computed in a pre-processing step.

3.3 Variant BKP-DP3

Variant BKP-DP3 follows the same principle of variant BKP-DP1 but the upper bound is computed in a different manner at a later stage. The main idea is to provide a tighter upper bound. At stage k, several solutions for a reduced $\{0, 1\}$ knapsack problem with the remaining $n-k$ objects that were not yet considered in the chosen partial solution are computed. In particular, two solutions are optimal for each objective of the reduced problem and the remaining ones are optimal for several weighted sum scalarized problems, by varying the weight parameters, or by using dichotomic search (see Section 3.1). Then, the profits of these solutions are summed up to the state $s \in T_k$, which provides several upper bounds. In the following, we consider only one optimal solution for the weighted sum scalarized problem with equal weights. We will denote by $u^1(s) := (u_1^1, u_2^1)$ and $u^2(s) := (u_1^2, u_2^2)$, the bounds that were obtained from optimal solutions for f^1 and f^2, respectively, and by $u^3(s) := (u_1^3, u_2^3)$, the bound obtained from the scalarized problem with equal weights.
The three upper bounds described above define two regions of interest in the objective space, denoted by A, which is delimited by $u^1(s)$, $u^3(s)$ and $(u_1^3 - u_2^1 + u_2^3, u_2^1)$, and B, which is delimited by $u^2(s)$, $u^3(s)$ and $(u_1^2, u_2^3 - u_1^2 + u_1^3)$. See left-hand-side plot of Figure 2. Note that A and B identify regions where upper bounds can still be found for a scalarized problem with a different combination of weights. However, region $f_1 - u_1^3 + f_2 - u_2^3 > 0$, denoted by C in the left plot of Figure 2, cannot have an upper bound by the optimality definition of $u^3(s)$. Note that, since there may be alternative optima for the scalarized problem, one should not expect that the regions A and B are connected, as shown in the right-hand-side plot of Figure 2.
The removal, or not, of states in T_k works as follows. For a given state $s \in T_k$ and the sequence $\left(\ell^1, \ldots, \ell^m\right)$ of m lower bounds generated as in variant BKP-DP1, sorted in increasing order of f_1 (and decreasing order of f_2), consider a monotone rectilinear polyline L that connects the points in this sequence. See the straight line in Figure 3. The state s can only be discarded if L and either region A or B do not intersect. If they intersect at any place, then s cannot be discarded. Figure 3 illustrates the latter case. The detection of whether L intersects a triangular region is performed by a classical technique of segment-segment intersection [5]. For efficient queries, balanced binary trees were used to implement both bound sequences.

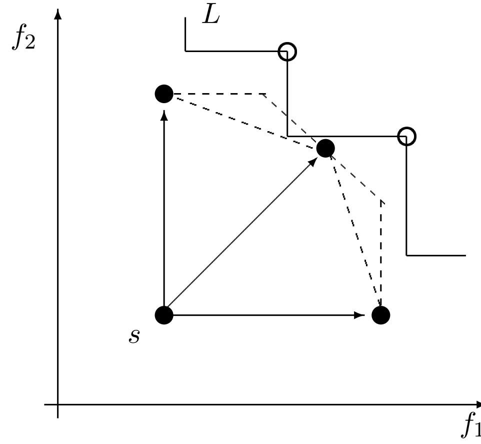

Figure 3: Illustration of a case in which state s is not discarded in BKP-DP3.

4 Computational experiments

4.1 Design of experiments

The experiments were performed in a computer cluster with 6 nodes, each with an AMD Phenom II X6 processor with 3.2GHz, 3 and 6 MB L2 and L3 Cache, respectively, and 12 GB DDR3 SDRAM. All implementations were written in C++ and shared the same data structures. The operating system was Ubuntu 8.04 LTS. All codes were compiled with g++ version 4.2.4 using the -O3 flag.
The bi-objective $\{0, 1\}$ knapsack instances were the same as described by Bazgan et al.[2]. Each weight and profit value of each instance is generated randomly according to an uniform distribution in a given range. Different ranges induces different structure on the input data that may affect algorithm performance. The instances considered are as follows:

- *Type A* (random instances) with weights and profits uniformly random generated in the range $[1, 1000]$.
- *Type B* (unconflicting instances) with profits v_j^1 and v_j^2 of each object j in the range $[111, 1000]$ and $[v_j^1 - 100, v_j^1 + 100]$, respectively; this induces a positive correlation between profits of an object; the weights are in the range $[1, 1000]$.
- *Type C* (conflicting instances) with the profit v_j^1 and weight w_j of each object j in the range $[1, 1000]$; the profit v_j^2 is in the range $[\max\{900 - v_j^1, 1\}, \min\{1100 - v_j^1, 1000\}]$, which induces a negative correlation between profits of an object.
- *Type D* (conflicting instances with correlated weights) Similar to Type C instances, but the weights are in the range $[v_j^1 + v_j^2 - 200, v_j^1 + v_j^2 + 200]$, which induces a positive correlation between the weight and the two profits of an object.

For every instance, the capacity W is half of the total sum of weight values.
Variant BKP-DP3 follows the same principle of variant BKP-DP1 until a given stage i. From this stage on, the algorithm computes an upper bound set of three solutions per state, as mentioned in Section 3.3. We considered $i = 0.05 \cdot n$, $0.08 \cdot n$ and $0.1 \cdot n$; we denote the corresponding variants by BKP-DP3$_{.05}$, BKP-DP3$_{.08}$ and BKP-DP3$_{.10}$, respectively. Preliminary experiments suggest that variant BKP-DP3 takes a large amount of time, if the computation of the new bound is performed at an earlier stage.

Type	n	$\lvert ND(Y)\rvert$	BKP-DP	BKP-DP1	BKP-DP2	BKP-DP3$_{.05}$	BKP-DP3$_{.08}$	BKP-DP3$_{.10}$
A	100	159.3	0.5	0.4	0.4	0.4	0.5	0.5
	200	529.0	26.6	19.7	19.8	21.4	20.7	20.3
	300	1 130.7	204.9	155.3	155.2	163.1	159.4	159.1
	400	1 713.3	835.9	603.5	600.9	630.4	617.2	627.6
	500	2 537.5	2 690.6	1 941.4	1 953.2	2 029.6	2 085.9	2 281.1
	600	3 593.9	6 862.8	5 052.7	5 019.6	5 282.3	5 663.6	6 433.1
	700	4 814.8	14 236.2	12 024.2	12 115.1	12 498.4	14 141.8	16 907.0
B	600	74.3	2.2	1.6	2.0	1.7	1.7	1.7
	700	78.6	3.6	2.6	3.2	2.7	2.7	2.7
	800	118.1	8.7	6.8	8.0	6.9	6.9	7.1
	900	124.4	11.0	8.7	10.6	8.8	9.1	9.3
	1 000	157.0	19.4	15.0	17.6	16.0	16.4	16.5
	2 000	477.7	624.5	487.5	514.9	493.9	502.3	510.6
	3 000	966.9	4 555.8	3 513.2	3 610.0	3 622.3	3 638.8	3 757.7
	4 000	1 542.3	17 676.0	15 442.0	15 829.0	15 767.9	15 866.5	16 336.4
C	100	558.2	5.5	4.2	4.3	4.7	4.5	4.4
	200	1 612.8	141.7	104.3	103.6	110.9	113.1	123.9
	300	2 893.6	1 027.3	731.4	724.0	783.8	904.5	1 076.1
	400	4 631.2	4 164.9	3 006.3	3 001.6	3 324.2	4 389.5	6 062.8
	500	7 112.1	9 324.7	8 625.4	9 910.0	8 642.3	14 892.4	21 357.6
D	100	1 765.4	93.5	82.8	83.1	88.5	89.9	91.1
	150	3 418.5	701.7	585.2	605.9	632.9	653.5	696.1
	200	5 464.0	3 246.9	2 792.3	2 750.3	2 943.3	3 166.5	3 652.0

Table 1: Average CPU-time of several approaches for the benchmark set of instances.

4.2 Results and discussion

Tables 1 and 2 presents the CPU-time and maximum number of states maintained in memory by the implementations of the original BKP-DP and the three variants proposed in this article. The values are averaged over each set of instances of the same type and size. The underlined values are the best for each type and instance size.
The experimental results indicate that variants BKP-DP1 and BKP-DP2 are able to reach an average of 20% improvement of CPU-time with respect to performance of BKP-DP as well as lower maximum number of states. Moreover, in most instances of type B, BKP-DP1 performs relatively faster than BKP-DP2. This fact seems to be related to the lower maximum number of states that are kept by BKP-DP1. However, in the remaining type of instances, the running time of BKP-DP1 and BKP-DP2 is very similar. Except for instances of type B, BKP-DP1 keeps the lowest maximum number of states. In general, variants BKP-DP3 take more time to terminate than variants BKP-DP1 and BKP-DP2, and even more time than BKP-DP on large instances of type A and C. The results also indicate that it is preferable to start the computation of the upper bound set in BKP-DP3 as latter as possible, which means that the time spent on computing a more precise upper bound does not pay-off the pruning. Still, the time spent by the best performing variant BKP-DP3 is always performing better than the original approach.

5 Conclusions

In this paper we presented some methodological and computational improvements for the state of the art implementation of dynamic programming based algorithms for bi-objective $0, 1$-knapsack problems. In general, all the proposed variants improved the baseline implementation, and the most performant ones led to an important gain in terms of CPU time and memory requirements, mainly in large instances; on average about 20% less time consuming and 15% less memory usage than the state of the art implementation. This shows that the classical and improved version of dynamic programming

Type	n	$\|ND(Y)\|$	BKP-DP	BKP-DP1,3$_*$	BKP-DP2
A	100	159.3	16 817	13 813	13 673
	200	529.0	206 910	165 805	164 385
	300	1 130.7	884 959	717 865	713 064
	400	1 713.3	2 157 218	1 693 984	1 683 407
	500	2 537.5	4 873 075	3 808 216	3 788 499
	600	3 593.9	9 486 857	7 630 158	7 590 283
	700	4 814.8	15 225 100	13 372 353	13 299 838
B	600	74.3	32 929	27 143	29 079
	700	78.6	42 408	34 790	37 404
	800	118.1	85 997	68 785	72 326
	900	124.4	88 199	72 202	76 885
	1 000	157.0	123 244	104 657	109 726
	2 000	477.7	1 448 870	1 191 158	1 209 903
	3 000	966.9	5 758 563	4 761 559	4 795 447
	4 000	1 542.3	14 674 987	13 151 485	13 211 354
C	100	558.2	103 412	79 711	77 457
	200	1 612.8	913 967	681 135	672 440
	300	2 893.6	3 446 587	2 589 395	2 554 477
	400	4 631.2	9 292 701	7 108 549	7 039 865
	500	7 112.1	13 440 388	13 309 231	13 204 383
D	100	1 765.4	1 129 540	1 037 776	1 021 365
	150	3 418.5	4 230 011	3 829 118	3 792 758
	200	5 464.0	12 347 355	11 167 991	11 090 087

Table 2: Maximum amount of states in memory of several approaches for the benchmark set of instances; BKP-DP1,3$_*$ denotes variants BKP-DP1, BKP-DP3$_{.05}$, BKP-DP3$_{.08}$ and BKP-DP3$_{.10}$.

algorithms can be improved with the insertion of non-classical techniques for improving the bound sets on the set of non-dominated solutions for the bi-objective case even for the most difficult instances available in the literature. More variants and combination of variants can be done in future research. One promising line of future research is the one related to variant BKP-DP3 and the computation of more than three solutions for deriving a bound. Another avenue for future research is to use the concept of bi-objective core and the heuristic associated with it to compute a good pool of initial solutions [7].

Acknowledgment J.R. Figueira, L. Paquete and D. Vanderpooten acknowledge the financial support from the Luso-French bilateral cooperation agreement between CEG-IST (FCT/CNRS 2009) and LAMSADE. J.R. Figueira and M. Simões acknowledge a CEG-IST grant from Instituto Superior Técnico (PTDC/GES/73853/2006). D. Vanderpooten acknowledges the support by the project ANR-09-BLAN-0361 “GUaranteed Efficiency for PAReto optimal solutions Determination (GUEPARD)”.

References

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

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

[3] Cristina Bazgan, Hadrien Hugot, and Daniel Vanderpooten. Implementing an efficient fptas for the 0-1 multi-objective knapsack problem. *European Journal of Operational Research*, 198(1):47–56, 2009.

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

[5] T.H. Cormen, C.E. Leiserson, R.K. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001.

[6] Thomas Erlebach, Hans Kellerer, and Ulrich Pferschy. Approximating multi-objective knapsack problems. In *WADS*, pages 210–221, 2001.

[7] C. Gomes da Silva, J. Clímaco, and J.R. Figueira. Core problems in bi-criteria 0,1-knapsack problems. *Computers and Operations Research*, 35(7):2292–2306, 2008.

[8] C. Gomes da Silva, J.C.N. Clímaco, and J.R. Figueira. A scatter search method for bi-criteria {0, 1}-knapsack problems. *European Journal of Operational Research*, 169(2):373–391, 2006.

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

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

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

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

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

[14] S. Martello and P. Toth. *Knapsack Problems – Algorithms and Computer Implementations*. John Wiley & Sons, 1990.

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

[16] D. Pisinger. A minimal algorithm for the 0-1 knapsack problem. *Operations Research*, 45:758–767, 1997.

[17] M.J. Rosenblatt and Z. Sinuany-Stern. Generating the discrete efficient fronteir to the capital budgeting problem. *Operations Research*, 37(3):384–394, 1989.

[18] R.E. Steuer. *Multiple Criteria Optimization: Theory, Computation and Application.* John Wiley, New York, 546 pp, 1986.

[19] J. Teng and G. Tzeng. A multiobjective programming approach for selecting non-independent transportation investiment alternatives. *Transportation Research-B*, 30(4):201–307, 1996.

[20] M. Visée, J. Teghem, M. Pirlot, and E. L. Ulungu. Two-phases method and branch and bound procedures to solve the bi–objective knapsack problem. *Journal of Global Optimization*, 12(2):139–155, 1998.