Using bound sets in multiobjective optimization: Application to the biobjective binary knapsack problem

Charles Delort, Olivier Spanjaard

Using bound sets in multiobjective optimization: Application to the biobjective binary knapsack problem

Charles Delort[1] and Olivier Spanjaard[1]

LIP6-CNRS, Université Pierre et Marie Curie (UPMC), 4 Place Jussieu
F-75005 Paris, France
{charles.delort,olivier.spanjaard}@lip6.fr

Abstract. This paper is devoted to a study of the impact of using bound sets in biobjective optimization. This notion, introduced by Villareal and Karwan [19], has been independently revisited by Ehrgott and Gandibleux [9], as well as by Sourd and Spanjaard [17]. The idea behind it is very general, and can therefore be adapted to a wide range of biobjective combinatorial problem. We focus here on the biobjective binary knapsack problem. We show that using bound sets in a two-phases approach [18] based on biobjective dynamic programming yields numerical results that outperform previous ones, both in execution times and memory requirements.

Keywords: Multiobjective combinatorial optimization, bound sets, biobjective binary knapsack problem.

1 Introduction

Multiobjective combinatorial optimization (MOCO) deals with combinatorial problems where every solution is evaluated according to several objectives. Interest in this area has tremendously grown over the last two decades. A thorough presentation of the field can be found for instance in a book by Ehrgott [7]. The standard approach aims at generating the whole set of Pareto optimal solutions, i.e. solutions that cannot be improved on one objective without being depreciated on another one. Most of the classical exact and approximate methods for finding an optimal solution in single objective discrete optimization have been revisited for finding the Pareto set under multiple objectives, e.g. dynamic programming [6, 12], branch and bound [4, 11, 14], greedy algorithm [16], as well as many heuristic and metaheuristic methods [8].

In order to perform implicit enumeration in multiobjective optimization problems, the formal notion of *bound set* needs to be introduced. This has been done several times in the literature. Roughly speaking, bound sets are *sets* of bounds. Indeed, due to the partial nature of the ordering relation between solutions, the use of a set of bounds instead of a single bound makes it possible to more tightly approximate the image set of the solutions in the objective space. To our knowledge, one of the first work mentioning that notion was performed by Villareal

and Karwan [19], and deals with branch and bounds for multiobjective integer linear programming problems. However, in this work and subsequent ones, no operational way to compute bound sets has been devised where the bound set does not reduce to a singleton. Very recently, based on the convex hull of the image of the solutions in the objective space, new bound sets have been proposed [9, 17]. The use of these new bound sets has proved very efficient in the biobjective spanning tree problem [17]. The purpose of the present paper is to show how these bound sets can be used to design efficient algorithms for the biobjective binary knapsack problem. Our contribution is twofold: we first explain how to hybridize multiobjective dynamic programming with the fathoming criterion provided by the bound sets, and then detail how multiobjective dynamic programming can be embedded in a two-phases approach to further improve the method. The hybridization we propose is in the spirit of the dominance relations between states used in a work by Bazgan *et al.* [2, 3], but enables huge savings in memory requirements as well as improvements in execution times. The two-phases version of the algorithm provides even better results thanks to a *shaving* procedure [13] that makes use of the bound sets.

2 Preliminaries

2.1 Preliminary definitions

We first recall some preliminary definitions concerning MOCO problems. They differ from the standard single objective ones mainly in their cost structure, as solutions are valued by m-vectors instead of scalars. Let us denote by $\mathcal{X}$ the set of feasible solutions, and by $\mathcal{Y}$ its image in the objective space. The image of solution $x \in \mathcal{X}$ is $f(x) = (f_1(x), \ldots, f_m(x))$. Comparing solutions in $\mathcal{X}$ amounts then to comparing m-vectors in $\mathcal{Y}$. In this framework, the following notions prove useful (in a maximisation setting):

Definition 1. *The* weak dominance *relation on m-vectors of $\mathbb{Z}_+^m$ is defined, for all $y, y' \in \mathbb{Z}_+^m$, by $y \succcurlyeq y' \iff [\forall i \in \{1, \ldots, m\}, y_i \geq y'_i)]$. The* dominance *relation is defined as the asymmetric part of $\succcurlyeq$: $y \succ y' \iff [y \succcurlyeq y'$ and $y' \not\succcurlyeq y]$.*

Definition 2. *Within a set $Y \subseteq \mathcal{Y}$, an element y is said to be* dominated *(resp.* weakly dominated*) when $y' \succ y$ (resp. $y' \succcurlyeq y$) for some y' in Y, and* non-dominated *when there is no y' in Y such that $y' \succ y$. The set of non-dominated elements in Y is denoted by $Y^\star$.*

By abuse of language, when $f(x) \succ f(x')$, we say that solution x *dominates* solution x'. Similarly, we use the term of *non-dominated* solutions. The set of non-dominated solution of $X \subseteq \mathcal{X}$ is denoted by $X^\star$. Following Bazgan *et al.* [2, 3], we say that a set of non-dominated solutions is *reduced* if it contains one and only one solution for each non-dominated objective vector in $Y = f(X) = \{f(x) : x \in X\}$. The aim of a multiobjective combinatorial problem is to determine a reduced set of non-dominated solutions.

2.2 Multiobjective binary knapsack problem

An instance of the multiobjective binary knapsack problem (0-1 MOKP) consists of a knapsack of integer capacity c, and a set of items $N = \{1, \ldots, n\}$. Each item j has a weight w^j and a m-vector profit $p^j = (p_1^j, \ldots, p_m^j)$, variables w^j, p_k^j ($k \in \{1, \ldots, m\}$) being integers. A solution is characterized by a binary n-vector x, where $x_j = 1$ if item j is selected. Furthermore, a solution x is feasible if it satisfies the constraint $\sum_{j=1}^{n} w^j x_j \leq c$. The goal of the problem is to find a reduced set of non-dominated solutions, which can be formally stated as follows:

$$\begin{aligned} \text{maximize } & \sum_{j=1}^{n} p_k^j x_j \qquad k \in \{1, \ldots, m\} \\ \text{subject to } & \sum_{j=1}^{n} w^j x_j \leq c \\ & x_j \in \{0, 1\} \quad j \in \{1, \ldots, n\} \end{aligned}$$

The special case when $k = 2$ is named *biobjective binary knapsack problem (0-1 BOKP)*.

Example 1. Consider the following problem:

$$\begin{aligned} \text{maximize } & \begin{cases} 10x_1 + 2x_2 + 6x_3 + 9x_4 + 12x_5 + x_6 \\ 2x_1 + 7x_2 + 6x_3 + 4x_4 + x_5 + 3x_6 \end{cases} \\ \text{subject to } & 4x_1 + 4x_2 + 5x_3 + 4x_4 + 3x_5 + 2x_6 \leq 6 \\ & x_j \in \{0, 1\} \quad j \in \{1, \ldots, 6\} \end{aligned}$$

The non-dominated solutions are: $\mathcal{X}^\star = \{(0,0,0,0,1,1), (1,0,0,0,0,1), (0,0,0,1,0,1), (0,1,0,0,0,1)\}$, and their image set in the objective space is $\mathcal{Y}^\star = \{(13,4), (11,5), (10,7), (3,10)\}$ (see Figure 1). Note that all solutions in $\mathcal{X}^\star$ have distinct images in the objective space, therefore $\mathcal{X}^\star$ is a reduced set of non-dominated solutions.

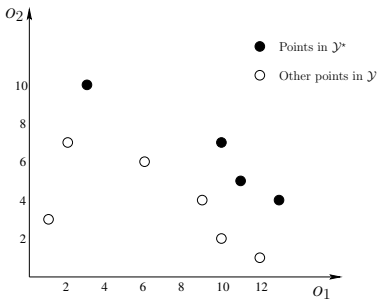

Fig. 1. Objective space.

Problem 0-1 MOKP can be solved by using a dynamic programming (DP) procedure. For the ease of presentation, we only detail here the way the non-dominated points in the objective space are computed. Note that the non-dominated solutions themselves can of course be recovered, by using standard bookkeeping techniques that do not impact on the computational complexity of the algorithm. Let subproblem $P(i, w)$ denote an instance of 0-1 MOKP consisting of item set $\{1, \ldots, i\}$, and capacity w. Let $Y(i, w)$ be the image set of the feasible solutions in $P(i, w)$. If all sets $Y^\star(i-1, w)$ are known, for $w \in \{0, \ldots, c\}$, then $Y^\star(i, w)$ can be computed by the recursive formula:

$$Y^\star(i, w) = \begin{cases} Y^\star(i-1, w) & \text{if } w < w^i \\ \text{ND}\left(Y^\star(i-1, w) \cup \{y + p^i : y \in Y^\star(i-1, w - w^i)\}\right) & \text{if } w \geq w^i \end{cases}$$

Notation ND$(\cdot)$ stands for a set function returning the subset of non-dominated points in a set of m-vectors. The complexity in time and space of the DP pro-

cedure crucially depends on the cardinality of sets $Y^\star(i,w)$. Any result enabling to discard elements in these sets is therefore worth investigating. Obviously, an element $y \in Y^\star(i,w)$ can be discarded if there exists an element $y' \in Y^\star(i,w')$ such that $w' < w$ and $y' \succcurlyeq y$. With the same goal in mind (discarding elements in the dynamic programming procedure), Villareal and Karwan presented a hybrid DP approach to solve multicriteria integer linear programming problems [19]. They hybridize DP with fathoming criteria and relaxations, so as to discard some elements that would not lead to non-dominated solutions. Since we use a similar technique (by providing a more powerful fathoming criterion), we are going to present and define the bound sets used to discard most of the unwanted elements.

3 Bound sets in MOCO problems

3.1 Definition of upper and lower bound sets

Having good upper and lower bounds is very important in many implicit enumeration methods. It is well known that the tightness of these bounds is a key parameter for the efficiency of the methods. In a multiobjective optimization setting, since one handles sets of m-vectors, the very notion of upper and lower bound has to be revisited. This work has been undertaken by Villareal and Karwan [19], by introducing the notion of *bound sets* (in the terminology of Ehrgott and Gandibleux [9]). Since the formalism used here slightly differs from the one presented in these works, we give below our own definitions of upper and lower bound sets.

Upper bound set. The simplest idea that comes to mind to upper bound a set Y of vectors is to define a single vector y^I such that $y^I_i = \max_{y \in Y} y_i$ for $i = 1, \ldots, m$. This point is called the *ideal point* of Y. However, this ideal point is usually very "far" from the points in Y. For this reason, it is useful to define an upper bound from a *set* of vectors instead of a singleton. Such a set is then called an *upper bound set* [9].

Definition 3 (upper bound set). *A set* $\mathbf{UB}$ *is an upper bound set of* Y *if* $\forall y \in Y,\ \exists u \in \mathbf{UB}\ : u \succcurlyeq y$.

This is compatible with the definition of an upper bound in the single objective case ($\mathbf{UB}$ reduces then to a singleton). As previously indicated, the upper bound set defined by $\mathbf{UB} = \{y^I\}$ is poor. In practice, a general family of good upper bound sets of Y can be defined as $\mathbf{UB}_\Lambda = \bigcap_{\lambda \in \Lambda} \{u \in \mathbb{R}^m : \langle \lambda, u \rangle \leq \mathbf{UB}_\lambda\}$, where the $\lambda \in \Lambda$ are weight vectors of the form $(\lambda_1, \ldots, \lambda_m) \geq 0$, $\langle ., . \rangle$ denotes the scalar product, and $\mathbf{UB}_\lambda \in \mathbb{R}$ is an upper bound for $\{\langle \lambda, y \rangle : y \in Y\}$. Of course, the larger $|\Lambda|$ is, the better the upper bound set becomes. Clearly, the best upper bound set in this family is obtained for $\Lambda = \Lambda_c(Y)$ where $\Lambda_c(Y)$ characterizes the facets of the non-dominated boundary of the convex hull of Y (see Example 2). Interestingly, we will see in the next subsection that this boundary can be efficiently computed in the biobjective case, provided $\mathbf{UB}_\lambda$ can be determined within polynomial or pseudo-polynomial time.

Lower bound set. Similarly to the upper bound set, the simplest idea that comes to mind to lower bound a set Y of vectors is to define a single vector y^A such that $y^A_i = \min_{y \in Y} y_i$ for $i = 1, \ldots, m$. This point is called the *anti-ideal point* of Y. Here again, taking several points simultaneously into account in the lower bound enables to bound more tightly set Y. Such a set is then called a *lower bound set* [9].

Definition 4 (lower bound set). *A set* $\mathbf{LB}$ *is a lower bound set of* Y *if* $\forall y \in Y,\ \exists l \in \mathbf{LB}\ : y \succcurlyeq l$.

As above, the compatibility with the single objective case holds. In the biobjective case, when Y only includes mutually non-dominated points, we will show in the next subsection a way to refine the lower bound set defined by $\mathbf{LB} = \{y^A\}$.

Comparing bound sets. Implicit enumeration is about eliminating entire subsets of solutions by using simple rules. In order to perform the elimination, we need to evaluate if a subset $X \subseteq \mathcal{X}$ of feasible solutions potentially includes non-dominated solutions in $\mathcal{X}$. To do this, one compares an upper bound set $\mathbf{UB}$ of $f(X)$ and a lower bound set $\mathbf{LB}$ of $f(\mathcal{X}^\star) = \mathcal{Y}^\star$. Unlike the single objective case, the comparison is not trivial since one handles sets instead of scalars. We introduce here two notions that make it possible to simply define this operation in a multiobjective setting.

Definition 5 (upper and lower relaxations). *Given an upper bound set* $\mathbf{UB}$*, the upper relaxation* $\mathbf{UB}^{\preccurlyeq}$ *is defined as:* $\mathbf{UB}^{\preccurlyeq} = \{x \in \mathbb{R}^m_+, \exists u \in \mathbf{UB},\ u \succcurlyeq x\}$. *Similarly, given a lower bound set* $\mathbf{LB}$*, the lower relaxation* $\mathbf{LB}^{\succcurlyeq}$ *is defined as:* $\mathbf{LB}^{\succcurlyeq} = \{x \in \mathbb{R}^m_+, \exists l \in \mathbf{LB},\ x \succcurlyeq l\}$.

Coming back to the comparison of $\mathbf{UB}$ and $\mathbf{LB}$, it is clear that $\mathbf{UB}^{\preccurlyeq} \supseteq f(X)$ and $\mathbf{LB}^{\succcurlyeq} \supseteq \mathcal{Y}^\star$. Consequently, $\mathbf{UB}^{\preccurlyeq} \cap \mathbf{LB}^{\succcurlyeq} = \emptyset$ implies that $f(X) \cap \mathcal{Y}^\star = \emptyset$. In this case, subset X can of course be safely pruned. Note that this pruning condition can be refined by using the fact that one only looks for a reduced set of non-dominated solutions as well as the fact that valuations are integers. Due to space constraints, this refinement is not detailed here. The main point is now to be able to efficiently compute good lower and upper bound sets. In the following subsection, this issue will be answered for the 0-1 BOKP.

3.2 Computation of bound sets in 0-1 BOKP

We now detail the algorithms used in 0-1 BOKP to compute the bound sets and perform their comparison.

Computation of an upper bound set. Given a subset $X \in \mathcal{X}$ of feasible solutions, upper bound set $\mathbf{UB}_{A_c(f(X))}$ can be compactly represented by storing the *extreme points* of $Y = f(X)$, i.e. the vertices of the non-dominated boundary of the convex hull of Y (points y^1, y^2, y^3, y^4 in the left part of Figure 2). Aneja and Nair's method [1] enables to efficiently compute these vertices in biobjective

combinatorial problems whose single objective version is solvable within polynomial or pseudo-polynomial time. It proceeds by launching a single objective version of the problem for determining each extreme points. The number of times the single objective solution method is launched is therefore linear in the number of extreme points.

Example 2. Let us come back to Example 1. Assume that one wants to upper bound the set $X_{\overline{6}}$ of feasible solutions where item 6 is not selected. Aneja and Nair's method yields the following list L of extreme points, characterizing $\mathbf{UB}_{\Lambda_c(f(X_{\overline{6}}))}$: $L = ((12,1),(9,4),(6,6),(2,7))$. The corresponding upper relaxation $\mathbf{UB}^{\preccurlyeq}_{\Lambda_c(f(X_{\overline{6}}))}$ is represented in Figure 2.

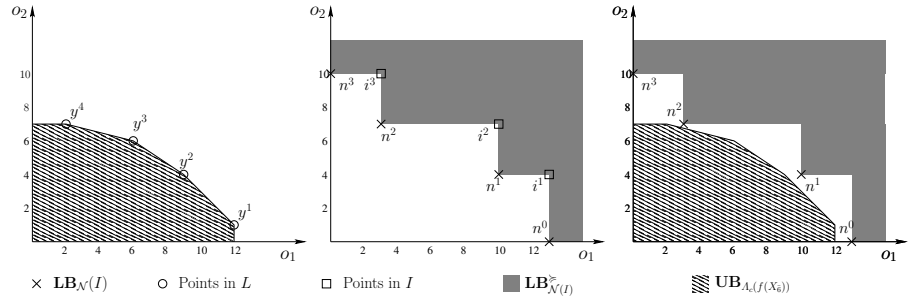

Fig. 2. Upper and lower bound sets in a biobjective setting.

Computation of a lower bound set. Given a subset $I \subseteq \mathcal{Y}$, a tight lower bound set $\mathbf{LB}$ of $I^\star$ can be computed as follows. When there are two objectives and $\{(i_1^j, i_2^j) : 1 \leq j \leq k\}$ are the points of $I^\star$ maintained in lexicographical order (i.e., in decreasing order of the first objective, and increasing order of the second one), one can set $\mathbf{LB}_{\mathcal{N}(I)} = \{n^j = (i_1^{(j+1)}, i_2^j) : 0 \leq j \leq k\}$, where $i_2^0 = 0$ and $i_1^{(k+1)} = 0$. The set $\mathbf{LB}_{\mathcal{N}(I)}$ can here be viewed as a generalization of the nadir point of I (whose components are the worst possible value among the points of $I^\star$). The points in $\mathbf{LB}_{\mathcal{N}(I)}$ are therefore sometimes called *local nadir points* [9]. One can note that $\mathbf{LB}_{\mathcal{N}(I)}$ is also a lower bound set for $\mathcal{Y}^\star$.

Example 3. Let us come back to Example 1 once again, and consider the following subset of points in $\mathcal{Y}$: $I = \{(13,4),(10,7),(3,10)\}$. The lower bound set is then: $\mathcal{N}(I) = \{(13,0),(10,4),(3,7),(0,10)\}$. This lower bound set is represented in the middle part of Figure 2, as well as its lower relaxation $\mathbf{LB}^{\succcurlyeq}_{\mathcal{N}(I)}$.

As described in the previous subsection, in order to know if one can prune a subset X of solutions, one must compute the intersection of the relaxations of a lower bound set of $\mathcal{Y}^\star$ and an upper bound set of $Y = f(X)$. Testing if $\mathbf{UB}^{\preccurlyeq}_{\Lambda_c(f(X))} \cap \mathbf{LB}^{\succcurlyeq}_{\mathcal{N}(I)} = \emptyset$ amounts to check whether one element of $\mathbf{LB}_{\mathcal{N}(I)}$ is included in $\mathbf{UB}^{\preccurlyeq}_{\Lambda_c(f(X))}$. It can be formally expressed by:

$$\forall n \in \mathbf{LB}_{\mathcal{N}(I)}, \exists \lambda \in \Lambda_c(f(X)) : \lambda_1 n_1 + \lambda_2 n_2 > \max_{y \in f(X)} (\lambda_1 y_1 + \lambda_2 y_2)$$

Example 4. Continuing Example 2 and Example 3, we shall compare the two obtained relaxations. Both sets are represented in the right part of Figure 2. Their intersection is empty, meaning that subset $X_{\bar{6}}$ can be safely discarded.

4 A new solution algorithm for 0-1 BOKP

Unlike the single objective case, in an implicit enumeration procedure for biobjective optimization, there is not a single incumbent but a *set* of incumbents: the set of non-dominated solutions among the solutions explored so far. For simplicity, we only refer to its image set $I \subseteq \mathcal{Y}$ in the following. The idea is then of course to discard subsets X of solutions such that $\mathbf{UB}^{\preccurlyeq}_{\Lambda_c(f(X))} \cap \mathbf{LB}^{\succcurlyeq}_{\mathcal{N}(I)} = \emptyset$. We now detail the various parts of our solution method for 0-1 BOKP.

4.1 Shaving procedure

The term "shaving" was introduced by Martin and Shmoys [13] for the job-shop scheduling problem. It enables to reduce the size of a problem by making some components forbidden or mandatory before starting the solution procedure. In knapsack problems, it amounts to consider subsets of solutions of the following form: for each item j, a subset X_j where item j is made mandatory, and a subset $X_{\bar{j}}$ where item j is made forbidden. For 0-1 BOKP, after initializing I with the extreme points of $\mathcal{Y}$ (by Aneja and Nair's method), the shaving procedure we propose consists in checking whether $\mathbf{UB}^{\preccurlyeq}_{\Lambda_c(f(X_j))} \cap \mathbf{LB}^{\succcurlyeq}_{\mathcal{N}(I)} = \emptyset$ or $\mathbf{UB}^{\preccurlyeq}_{\Lambda_c(f(X_{\bar{j}}))} \cap \mathbf{LB}^{\succcurlyeq}_{\mathcal{N}(I)} = \emptyset$. If X_j or $X_{\bar{j}}$ grants no non-dominated solution in $\mathcal{X}$, item j can be excluded from the problem by permanently setting $x_j = 0$ or $x_j = 1$. Note that the computation of the upper bound sets yields feasible solutions, possibly non-dominated. Consequently, during the running of the shaving procedure, set I is updated by inserting these possible new non-dominated elements. The shaving procedure is therefore launched twice in order to exclude some additional items during the second round of the procedure. Example 4 above shows that it is possible to shave item 6 in Example 1, by setting $x_6 = 1$.

4.2 Hybrid dynamic programming

During the dynamic programming (DP) procedure, the use of bound sets as a fathoming criterion, makes it possible to considerably reduce the number of stored elements in each set $Y^\star(i, w)$. This is called *hybridization*. Given an element $y \in Y^\star(i, w)$, by abuse of notation, we denote by $f^{-1}(y)$ a feasible solution in $P(i, w)$ such that $f(f^{-1}(y)) = y$ (if there are several solutions with the same image in the objective space, $f^{-1}(y)$ is any of them), and we denote by $X_y \subseteq \mathcal{X}$ the subset of feasible solutions in $P(n, c)$ whose projection on $P(i, w)$ is $f^{-1}(y)$. When computing $Y^\star(i, w)$ by DP, the fathoming criterion consists in discarding any element y such that $\mathbf{UB}^{\preccurlyeq}_{\Lambda_c(f(X_y))} \cap \mathbf{LB}^{\succcurlyeq}_{\mathcal{N}(I)} = \emptyset$. Finding $\mathbf{UB}_{\Lambda_c(f(X_y))}$ can be done by applying Aneja and Nair's method to find the extreme points of the

subproblem on $\{i+1,\ldots,n\}$ with capacity $c-w$, that is denoted by $\bar{P}_{(i+1,w)}$:

$$\text{maximize} \sum_{j=i+1}^{n} p_k^j x_j \qquad k \in \{1,2\}$$

$$\text{subject to} \sum_{j=i+1}^{n} w^j x_j \leq c-w \qquad x_j \in \{0,1\}$$

One can then obtain the vertices of $\mathbf{UB}_{\Lambda_c(f(X_y))}$ by simply translating the extreme points of $\bar{P}_{(i+1,w)}$ by y.

4.3 Two-phases method

Visée *et al.* [20] introduced a *two-phases method* to solve the biobjective binary knapsack problem. They first calculate the set of *extreme solutions* (i.e., whose images in the objective space are extreme points of $\mathcal{Y}$), and second, by launching several branch-and-bound procedures, they compute the set of non-extreme non-dominated solutions located in the triangles generated in the objective space by two successive extreme solutions. Since the work of Visée *et al.*, other approaches have been proposed that outperform the two-phases method: a labeling approach developed by Captivo *et al.* [5], and the already mentioned DP approach by Bazgan *et al.* [2, 3]. We propose here a two-phases version of our DP procedure. This technique is called *two-phasification* in the sequel. Instead of applying one single DP procedure directly on the 0-1 BOKP instance, one first computes the extreme solutions, and then applies one DP procedure for each triangle T in the objective space. Let us denote by $Y_T \subseteq \mathbb{R}^m$ the subset of the objective space corresponding to triangle T. When applying the DP procedure for finding feasible solutions within T, one checks whether $\mathbf{UB}^{\preccurlyeq}_{\Lambda_c(f(X_j))} \cap \mathbf{LB}^{\succcurlyeq}_{\mathcal{N}(I)} \cap Y_T = \emptyset$ during the local shaving procedure, and one checks whether $\mathbf{UB}^{\preccurlyeq}_{\Lambda_c(f(X_y))} \cap \mathbf{LB}^{\succcurlyeq}_{\mathcal{N}(I)} \cap Y_T = \emptyset$ for the fathoming criterion. Clearly, these conditions will hold much more frequently than if the problem is considered in its whole. Moreover, one can limit the computation of the upper bound sets to the area of the triangle under consideration. By subdividing the problem in this way, both the shaving procedure and the fathoming criterion are more efficient, since one focuses on a restricted area of the objective space. This is confirmed by the numerical experiments.

Example 5. In Figure 3 are represented the triangles that would be obtained in the problem described in Example 1. In a two-phases method, the feasible solutions corresponding to the extreme points (in black) would be found during the first phase, and the other non-dominated solutions (grey points) would be found during the second phase.

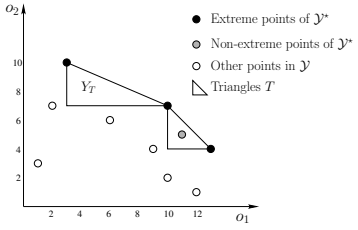

Fig. 3. Two-phases method.

5 Numerical experiments

All experiments presented here were performed on an Intel® Core™ 2 Duo CPU E8400 @ 3.00GHz personal computer, endowed with 3.2GB of RAM memory. All algorithms were written in C++. To solve the single objective knapsack problems, we used the minknap algorithm [15] which proved to be one of the quickest in the literature (see the book by Kellerer *et al.* [10]).

5.1 Instances

The types of instances considered here are the same as in [2, 3], where the parameters are uniformly randomly generated and $c = \lceil 0.5 \sum_{j=1}^{n} w^j \rceil$.
Type A: random instances, where p_1^j, p_2^j and $w^j \in \{1, \ldots, 1000\}$;
Type B: unconflicting instances, where $p_1^j \in \{101, \ldots, 1000\}$, $p_2^j \in \{p_1^j - 100, \ldots, p_1^j + 100\}$ and $w^j \in \{1, \ldots, 1000\}$;
Type C: conflicting instances, where $p_1^j \in \{1, \ldots, 1000\}$, $p_2^j \in \{\max\{900 - p_1^j, 1\}, \ldots, \min\{1100 - p_1^j, 1000\}\}$ and $w^j \in \{1, \ldots, 1000\}$;
Type D: conflicting instances with correlated weights, where $p_1^j \in \{1, \ldots, 1000\}$, $p_2^j \in \{\max\{900 - p_1^j, 1\}, \ldots, \min\{1100 - p_1^j, 1000\}\}$ and $w^j \in \{p_1^j + p_2^j - 200, \ldots, p_1^j + p_2^j + 200\}$.

5.2 Results

We compared our method (named S2H for Shaving, 2-phases, and Hybrid DP) and the one of Bazgan *et al.* [2, 3] (named BHV: initials of the authors) by running both methods on the same instances[1]. Table 1 shows the time and memory spent to solve different types and sizes of instances. The first two columns indicate the size and type of the instances solved. For each size and type, 30 randomly generated instances have been solved using different methods, and the average and maximum times and memory requirements are indicated. Numbers in bold represent the best value for a given type and size. Shaving, hybridization and two-phasification are the three main parts of the algorithm presented in this paper. We evaluated some variations of our method in order to measure the importance of each part: 2H is a two-phases method using a hybridized DP procedure, SH is a hybridized DP procedure applied to a shaved problem, and finally S2 is a two-phases method using simple DP on problems reduced by shaving. A time limit was set to 10000 seconds. Symbol "-" in the table denotes that at least one instance of this type and size reached this limit. Symbol "*" indicates that at least one instance couldn't be solved due to insufficient memory.

Shaving. The shaving procedure is particularly effective on instances of type A or B: there is indeed a lot of items that are not interesting, such as items with low profits on both objectives, and high weights, and conversely items that have

[1] We wish to thank Hadrien Hugot who kindly sent us the code of the BHV method.

high profits and low weights, which will be taken in all non-dominated solutions. On the other hand, since all items in types C and D have conflicting profits, it is more difficult to shave items.

Two-phases. Using a two-phases method makes it possible to divide the problem into several smaller problems, but there can be a lot of them (for problems of type A and size 1000, there are on average 155 subproblems to solve). The combination with the shaving procedure is interesting, because it further reduces the sizes of the subproblems. The memory space spared this way is not as important as that of the shaving for type A and B; the opposite is observed for types C and D.

Hybridization. Hybridizing the DP is the main part of our algorithm. Not only does it tremendously reduces the memory requirements, it also saves a lot of computation time for larger, or more difficult instances. This can be seen by looking at the results of method S2 on instances of types C and D, both in terms of time and memory requirements.

We will now compare the S2H method to the BHV method. First, from a memory consumption point of view, our method largely outperforms the BHV method for all sizes and types of instances. From the computation time perspective, results depend on the sizes and types of instances. For types A and B the S2H method is much faster than the BHV method, while for types C and D it is slower, although when the size grows our method seems to become more and more competitive (it is as good as method BHV for type C and size 500, and within a factor two for type D and size 250). The reason for this behaviour is that the fathoming criterion is rather time consuming, but this is compensated for bigger instances by the fact that a lot of computation time is saved thanks to the important number of elements that have been fathomed.

6 Conclusion

In this paper, we have presented a new solution algorithm for 0-1 BOKP, based on the use of bound sets. It outperforms previous dynamic programming approaches from the viewpoint of memory requirements. Concerning the resolution times, the performances are better than the best known algorithm for this problem on random and unconflicting instances, and slower on conflicting instances (but within the same order of magnitude). A natural extension of this work would be to investigate the impact of the use of bound sets on other MOCO problems. Another extension would be to study how to improve the resolution times on conflicting instances of 0-1 BOKP. For this purpose, an incremental resolution of the single objective problems is worth investigating. Finally, note that our fathoming criterion has only been implemented in the biojective case up to now. The study of its practical implementation in problems involving more than two objectives is an interesting and potentially fruitful task in our opinion.

Acknowledgements. This work was supported by ANR project GUEPARD.

Type	Size	Method	Time (sec.)		RAM (MB)		Type	Size	Method	Time (sec.)		RAM (MB)	
			Avg.	Max.	Avg.	Max.				Avg.	Max.	Avg.	Max.
A	300	S2H	**17**	**30**	**2.6**	**3.1**	B	1000	S2H	7.0	11	**2.0**	**2.3**
		BHV	51	103	80	113			BHV	**4.1**	**10**	11	15
		2H	73	134	5.8	6.3			2H	17	40	15.2	16
		SH	60	88	4.0	4.4			SH	10	16	2.5	2.8
		S2	113	208	13.4	16			S2	10	15	2.3	3.6
	500	S2H	**73**	**109**	**3.1**	**3.7**		2000	S2H	**50**	**68**	**2.6**	**3.0**
		BHV	564	1031	401	449			BHV	132	272	132	272
		2H	459	626	8.8	9.3			2H	195	334	30.1	31
		SH	448	679	6.2	6.6			SH	109	181	3.8	4.3
		S2	671	1045	38	56			S2	91	123	14	21
	700	S2H	**193**	**254**	**3.6**	**3.8**		3000	S2H	**160**	**211**	**3.1**	**3.4**
		BHV	2740	4184	1308	1800			BHV	874	1292	449	449
		2H	1566	2038	11.1	12			2H	830	1227	44.6	45
		SH	2209	3353	8.7	9.4			SH	517	699	4.9	5.2
		S2	2820	3624	116	159			S2	344	468	45	74
	1000	S2H	**558**	**705**	**4.2**	**4.7**		4000	S2H	**358**	**435**	**3.7**	**3.9**
		BHV	*	*	*	*			BHV	3017	4184	1307	1800
		2H	5588	6981	15.7	16			2H	2292	3032	59.1	60
		SH	-	-	-	-			SH	1648	2097	6.1	6.4
		S2	-	-	-	-			S2	970	1308	84	130
C	200	S2H	73	121	4.3	5.0	D	100	S2H	84	136	**5.1**	**6.0**
		BHV	**32**	**47**	63	113			BHV	**35**	57	80	113
		2H	112	172	4.8	5.3			2H	108	169	5.1	6.0
		SH	147	239	**3.1**	**3.4**			SH	125	165	6.0	6.7
		S2	1835	2307	107	163			S2	2138	3252	124	168
	300	S2H	319	497	**5.9**	**6.9**		150	S2H	389	723	**7.5**	**8.8**
		BHV	**206**	**288**	257	449			BHV	**154**	**228**	311	449
		2H	539	832	6.7	7.3			2H	517	879	7.5	8.8
		SH	788	1159	9.4	10			SH	698	1123	9.2	10
		S2	-	-	-	-			S2	-	-	-	-
	400	S2H	946	1479	**7.7**	**9.0**		200	S2H	1143	2015	9.7	**12**
		BHV	**748**	**1006**	782	897			BHV	**770**	897	897	897
		2H	1756	2647	8.9	9.9			2H	1596	2796	**9.5**	12
		SH	2806	3956	14.8	18			SH	2689	3747	13.1	16
		S2	-	-	-	-			S2	-	-	-	-
	500	S2H	2138	3046	**9.6**	**10**		250	S2H	2555	3540	**11.7**	**17**
		BHV	**2014**	**2651**	1458	1800			BHV	**1989**	1100	1730	1800
		2H	4165	5952	10.4	11			2H	3585	4668	11.7	17
		SH	-	-	-	-			SH	6984	8516	18.1	21
		S2	-	-	-	-			S2	-	-	-	-

2H: method S2H without shaving SH: method S2H without two-phases
S2: method S2H without hybridization

Table 1. Computation times, and memory requirements of different methods for the 0-1 BOKP.

References

1. Y.R. Aneja and K.P.K. Nair. Bicriteria transportation problem. *Management Science*, 25:73–78, 1979.
2. C. Bazgan, H. Hugot, and D. Vanderpooten. An efficient implementation for the 0-1 multi-objective knapsack problem. In *WEA*, pages 406–419, 2007.
3. C. Bazgan, H. Hugot, and D. Vanderpooten. Solving efficiently the 0-1 multi-objective knapsack problem. *Computers & Operations Research*, 36(1):260–279, 2009.
4. G. Bitran and J.M. Rivera. A combined approach to solve binary multicriteria problems. *Naval Research Logistics Quarterly*, 29:181–201, 1982.
5. M.E. Captivo, J. Clìmaco, J. Figueira, E. Martins, and J.L. Santos. Solving bicriteria 0-1 knapsack problems using a labeling algorithm. *Computers & Operations Research*, 30(12):1865–1886, 2003.
6. H.G. Daellenbach and C.A. De Kluyver. Note on multiple objective dynamic programming. *Journal of the Operational Research Society*, 31:591–594, 1980.
7. M. Ehrgott. *Multicriteria Optimization, second edition*. Springer, 2005.
8. M. Ehrgott and X. Gandibleux. Approximative solution methods for multiobjective combinatorial optimization. *Journal of the Spanish Statistical and Operations Research Society*, 12(1):1–88, 2004.
9. M. Ehrgott and X. Gandibleux. Bound sets for biobjective combinatorial optimization problems. *Computers & Operations Research*, 34(9):2674–2694, 2007.
10. H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, Berlin, Germany, 2004.
11. G. Kiziltan and E. Yucaoglu. An algorithm for multiobjective zero-one linear programming. *Management Science*, 29(12):1444–1453, 1983.
12. K. Klamroth and M.M. Wiecek. Dynamic programming approaches to the multiple criteria knapsack problem. *Naval Research Logistics*, 47:57–76, 2000.
13. P.D. Martin and D.B. Shmoys. A new approach to computing optimal schedules for the job shop scheduling problem. In *Proceedings of the Fifth international IPCO conference, Vancouver, Canada*, pages 389–403. LNCS 1084, 1996.
14. G. Mavrotas and D. Diakoulaki. A branch and bound algorithm for mixed zero-one multiple objective linear programming. *European Journal of Operational Research*, 107:530–541, 1998.
15. D. Pisinger. A minimal algorithm for the 0-1 knapsack problem. *Operations Research*, 45:758–767, 1997.
16. P. Serafini. Some considerations about computational complexity for multiobjective combinatorial problems. In *Recent advances and historical development of vector optimization*, LNEMS 294, 1986.
17. F. Sourd and O. Spanjaard. A multi-objective branch-and-bound framework. Application to the bi-objective spanning tree problem. *INFORMS Journal of Computing*, 20(3):472–484, 2008.
18. B. Ulungu and J. Teghem. The two-phase method: An efficient procedure to solve bi-objective combinatorial optimization problems. *Foundations of Computing and Decision Sciences*, 20(2):149–165, 1995.
19. B. Villareal and M.H. Karwan. Multicriteria integer programming: A (hybrid) dynamic programming recursive approach. *Mathematical Programming*, 21:204–223, 1981.
20. M. Visée, J. Teghem, M. Pirlot, and B. Ulungu. Two-phases method and branch and bound procedures to solve the bi–objective knapsack problem. *J. of Global Optimization*, 12(2):139–155, 1998.