Multi-objective Branch and Bound for Mixed 0-1 Linear Programming : Corrections and Improvements for the Biobjective Case

Thomas Vincent

AG OPTIMIERUNG
Universität Kaiserslautern
Postfach 3049
67653 Kaiserslautern – Germany
thomas.vincent@etu.univ-nantes.fr

Abstract. This work addresses the correction and improvement of Mavrotas and Diakoulaki's branch and bound algorithm for mixed 0-1 MOLP. We first develop about the issues encountered by the original algorithm and propose and corrected version for the biobjective case. We then introduce several improvements about the bounds and end with the proposition of a new algorithm based on the two phase method and the Branch and Bound.

1 Introduction

We introduce the main definitions, properties and notations of multi-objective linear programming and multi-objective integer linear programming. The interested reader can find a more thorough introduction in [7] and [16].

A *multiple objective linear program* (MOLP) can be formulated as follows:

$$\begin{array}{ll} \min & Cx \\ \text{s.t.} & x \in X \end{array}$$

Here, $C = (c^1, ..., c^p)^T$ with rows $c^1, ..., c^p$ denotes a $p \times n$ linear objective matrix, $x \in \mathbb{R}^n$ the vector of variables, and

$$X := \{x \in \mathbb{R}^n : Ax \leqq b, x \geqq 0\}$$

is the *feasible set* in decision space $\mathbb{R}^n$. A is a $m \times n$ matrix of constraints and $b \in \mathbb{R}^m$ the right hand side vector. We refer to

$$Y := CX := \{y := Cx \in \mathbb{R}^p : x \in X\}$$

as the *outcome set* in objective space $\mathbb{R}^p$.

By adding integrality requirements to the variables in X we obtain a *multiple objective integer linear program* (MOILP). MOLP and MOILP are special cases of *Multiple Objective Mixed Integer Linear Program* (MOMILP) where the

integrality requirements are only added to a subset of the variables in X. For all these problems we assume that the objective functions are conflicting. This assumption guarantees that there does not exist an *ideal solution* $x \in X$ which minimizes simultaneously all p objectives.

As there is no canonical ordering defined in $\mathbb{R}^p$ for $p \geq 2$, minimizing a vector-valued objective function needs to be explained. We use the Pareto concept of optimality which is based on the three binary relations $\leqq$, $\leq$ and $<$. Let $y^1, y2 \in \mathbb{R}^p$. Then

$$y^1 \leqq y^2 \Leftrightarrow y^1_k \leq y^2_k \; \forall k = 1, ..., p$$
$$y^1 < y^2 \Leftrightarrow y^1_k < y^2_k \; \forall k = 1, ..., p$$
$$y^1 \leq y^2 \Leftrightarrow y^1 \leqq y^2 \text{ and } y^1 \neq y^2$$

A point $y^1 \in \mathbb{R}^p$ *dominates* (resp. *strictly dominates*, *weakly dominates*) $y^2 \in \mathbb{R}^p$ if $y^1 \leq y^2$ (resp. $y^1 < y^2$, $y^1 \leqq y^2$). The *Pareto cone* is defined as $\mathbb{R}^p_{\geqq} := \{y \in \mathbb{R}^p : y \geqq 0\}$.

A feasible solution $\hat{x} \in X$ is called efficient (or Pareto optimal) if there does not exist $x \in X$ such that $Cx \leq C\hat{x}$. In other word, no solution is at least as good as $\hat{x}$ for all objective functions and strictly better for at least one. When $\hat{x}$ is efficient, $C\hat{x}$ is called *non-dominated*. The *efficient set* $X_E \subseteq X$ is defined as

$$X_E := \{x \in X : \nexists \bar{x} \in X : C\bar{x} \leq Cx\}$$

and its image under the vector-valued linear mapping C is the *non-dominated set* $Y_N := CX_E$. The goal in an exact solution of a MOLP, MOILP or MOMILP is to determine a complete set, i.e. a set containing at least one efficient solution for each non-dominated point in Y_N.

The *efficient frontier* (or *Pareto frontier*) is defined as the set $\{y \in \text{conv}(Y_N) : \text{conv}(Y_N) \cap (y + (-\mathbb{R}^p_{\geqq})) = y\}$ where conv is the convex hull operator. For MOLP the efficient frontier is identical with Y_N and in the case of $p = 2$ objectives, the efficient frontier is known to be piecewise linear and convex. Its breakpoints are the *extreme non-dominated points* which are images of some of the *extreme efficient solutions* of decision space. If a non-dominated objective vector is on the efficient frontier it is called a *supported* non-dominated vector. Otherwise it is an *unsupported* non-dominated objective vector. The corresponding solutions in decision space are called *supported efficient solutions* and *unsupported efficient solutions*. It is important to notice that for MOLP every efficient solutions are supported, whereas for MOILP unsupported efficient solutions may exist.

Each supported efficient solution can be found as an optimal solution to the *weighted sum problem*, $\min\{\lambda Cx, x \in X\}$, for a certain $\lambda \in]0, 1[^p$ [9,10].

Two different notions of connectivity respectively based on topology and graph theory are used in the context of multiple objective programming.

We call a set S *topologically connected* if there does not exist non-empty open sets S_1 and S_2 such that $S \subseteq S_1 \cup s_2$ and $S_1 \cap S_2 = \emptyset$. For MOLP the efficient set X_E and the efficient frontier Y_N are topologically connected and Y_N is composed

of faces of dimension 0 to $p-1$. In contrast, neither X_E nor Y_N are topologically connected for MOILP.

Let $G = (N, A)$ denote the *adjacency graph of MOLP* where N is the set of efficient basic feasible solutions and A is the set of edges between nodes which can be obtained from each other by a single pivot operation. Isermann showed that this adjacency graph is connected (see [11]). It is therefore possible to find X_E and Y_N by simple pivot exchange arguments for MOLPs. Multiple objective simplex algorithm are based on this graph connectivity property and allow to find all the extreme efficient solutions and in an implicit manner all the efficient faces in the decision space and all the non-dominated faces in the objective space. However, many authors consider extreme non-dominated points in the outcome set as sufficient information on the solution of the problem, and efficient methods have been designed for this purpose. See [1,2] for discussions on this topic.

Literature about MOIP problems can be found in [5,18]. See [3,4,6,14] for literature about MOIP problems with binary variables only.

The important differences between MOLP and MOILP problems beg the question of the properties of efficient and non-dominated sets for MOMILPs. Are they topologically like in MOLP problems or not, like in MOILP problems ? Are these sets graph connected ? As MOMILP shares properties with both MOLP and MOILP, everything is likely to happen, especially efficient and non-dominated sets partially connected.

The rest of the paper is organized as follows. In Section 2 we describe the original multiple objective branch and bound algorithm and discuss about some issues and how to fix them. Then we present several improvements for the branch and bound in Section 3 and computational results in Section 4.

2 Mavrotas and Diakoulaki's multiple objective branch and bound

The branch and bound algorithm for mixed binary multiple objective linear programming presented by Mavrotas and Diakoulaki in [12,13] is a modified version of the conventional single objective branch and bound. The conventional algorithm systematically examines all the possible combinations of discrete variables in order to find the optimum solution of the single objective problem but is not designed for multiple objective problems where more than one solution can be efficient.

2.1 Definitions

Here we define the terms used to describe the procedure. They are mainly reused from [12].

The *combinatorial tree* is the structure built to examine all possible solutions to the problem. During its construction the binary variables are sequentially assigned to 0 and 1. Until a binary variable is assigned to a value, it is called

a *free variable* and considered as a linear variable varying in the interval [0,1]. Like in a single-objective Branch and Bound, the linear relaxation of the problem with fixed variables will be considered at each node, i.e. a MOLP. Since each intermediate node of the tree will have exactly two successors (or children), we can call it *binary tree*.

A *node* of the combinatorial tree is a partial solution of the problem. A subset of the binary variables are already assigned to 0 or 1. The *root node* is the first node of the tree and corresponds to the linear relaxation of the problem without fixed variable of the problem where every binary variables are free. A *final node* is a leaf node, meaning that all the binary variables are assigned to a value.

A *branch* of the tree is a sequence of linked nodes.

Fathoming a node means to stop the exploration of the branch corresponding to the node because it is established that none of its children will lead to interesting solutions. This occurs when the current node is already infeasible or is dominated by one or more final nodes.

Partially non-dominated points are the non-dominated points generated from each final node. It is important to declare a solution "partially" efficient until all the final nodes are explored. Such a solution indeed corresponds to an efficient solution of a sub problem and we have no guarantee about its efficiency for the general problem. The partially efficient solutions are only candidates for being efficient for the general MOMILP problem. They are stored in a database D_{ex} during the exploration of the binary tree.

Pseudo non-dominated points are the extreme points only dominated by a convex combinations of two other extreme partially non-dominated points.

Non-dominated points are points that are proved to be non-dominated for the general problem. They are the former partially efficient points which are still in D_{ex} at the end of the procedure.

Efficient combinations are the combinations of values of the 0-1 variables which lead to efficient solutions. They point out the branches which end to final nodes where efficient points are generated.

2.2 Description of the algorithm

The branch and bound algorithm is a depth first search in the combinatorial tree. It starts from the root node and moves downward to the final nodes. When one of them is reached, the corresponding MOLP problem is solved and its non-dominated points are computed. When a node is found to be fathomed, the algorithm backtracks to evaluate the remaining branches. When every possible combination of the binary variables has been examined, the process stops and D_{ex} contains the efficient points.

When a node is explored, the corresponding MOLP model is created from the original problem and the current partition of binary variables given by the branch. The assigned variables are implicitly evaluated, i.e. they are replaced by their value (0 or 1) in the constraints and the objective functions. For each free

variable an inequality constraint is added to the model to bound the variable in the interval $[0, 1]$. The ideal vector of the local problem is calculated by individual optimization of each objective function and is then compared to the points stored in D_{ex}. If there exists a point dominating the ideal vector, then every feasible solution of the local problem is dominated by the same point. In this case or if the problem is infeasible, the node is fathomed.

Each time a leaf node is considered, the partially non-dominated points are generated and D_{ex} is possibly updated. It is important to notice that here the partially non-dominated points are only the extreme non-dominated points of the MOLP. Consequently, the update of D_{ex} is done by the classical pairwise comparison merging two sets of points inside of which no point dominates another one.
Mavrotas and Diakoulaki (2005) have proposed an improved version of their algorithm. In particular, they have noticed that non-dominated points of MOLP are not reduced to extreme non-dominated points, and that consequently their first algorithm may generated dominated points. Thus, they have added a final filtering test in their improved algorithm.
On Figure 1 two extreme efficient points oD_1 and D_2 generated from the same final node belong to D_{ex}. D' is a convex combination of them and dominates the partially-efficient point of D_3 without B_1 nor B_2 dominating it so D_3 has to be removed from D_{ex}. In order to handle this case, Mavrotas and Diakoulaki proposed a test consisting, for each point of D_{ex}, in checking dominance with each feasible combination of two partially efficient points.

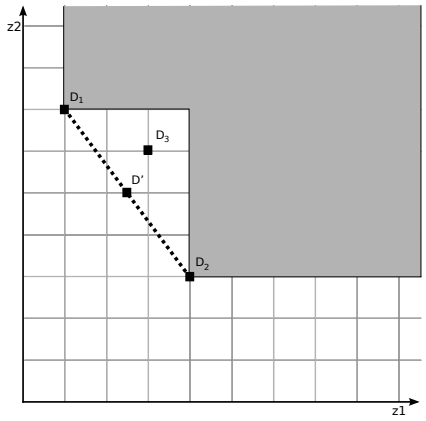

Fig. 1. Dominated point of D_{ex} now detected.

3 Corrections and improvements

3.1 Missed dominated and non-dominated solutions

As it is done very often for MOLP, Mavrotas and Diakoulaki's method stores only extreme non-dominated points, but every pseudo-efficient solutions may be not filtered. The process for the filtering of pseudo-efficient solutions is disconnected of the main test and nothing is said about when it is applied. We can consider three possible cases :

- The efficient extreme points of each efficient combination and their efficient edges are deduced from D_{ex} after the termination of the branch and bound algorithm. However, some extreme points of these efficient combinations may have been filtered and consequently some efficient edges loosed.
- The efficient extreme points of each efficient combination have been stored during the solution, therefore every efficient solutions should be stored. Unfortunately, even in non-efficient combinations (i.e. combinations of extreme points that have been all filtered), some part of edges may be efficient and therefore useful to filter pseudo-efficient points in D_{ex}.
- The efficient extreme points of each combination (efficient or not) are stored for the final filtering. Indeed, we do not know immediately if a combination is efficient so it may be natural to store it, as partially efficient extreme points are stored. In this case, no efficient points are lost but this solution is heavy in memory and requires so many dominance test (considering efficient and non-efficient combinations) that it is not realistic to apply it.

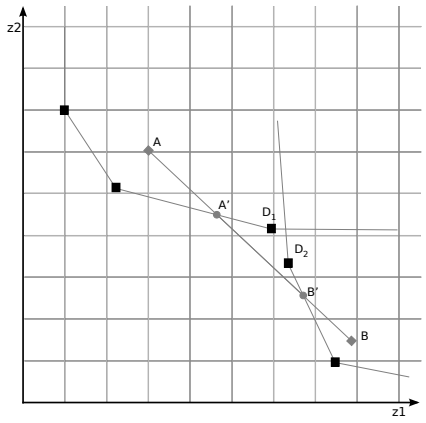

Fig. 2. Pseudo-efficient points of D_{ex} still not detected.

The second case is shown in Figure 2. In this example, D_1 and D_2 will be kept in D_{ex} although there are dominated by convex combinations of points A

and B. The dominance test fails because A and B are dominated so they are not stored in D_{ex} and the information about the efficient edge $[A', B']$ is lost. Extreme efficient points are consequently not enough to keep all the information needed to describe the non-dominated set of the main problem and we cannot afford to store every pseudo-efficient points in order to perform a very large number of dominance tests. It is therefore necessary to also store points like A' and B' on Figure 2 that are actually non-extreme points but keep track of the non-dominated faces and allow to not keep track of pseudo-efficient points like A and B that are not relevant anymore. The example used represents a biobjective problem but the issues are obviously the same for $p \geq 2$. In the rest of this paper, we will consider the biobjective case only due to the greater easiness to represent faces (points and edges).

We propose to store pseudo-efficient points and guide points in an ordered set called S_N instead of D_{ex} since we no longer consider only extreme non-dominated points. In addition to this, each point of S_N is associated with three flags. The two first flags respectively indicate whether or not the point is connected with its predecessor and successor in the list. From now "connected" will refer to the notion of "graph connection" presented in the first section, that is two points connected necessarily come from the same MOLP. The third flag is set on *true* when the corresponding point is an extreme efficient point.

Thanks to this representation, we can build Y_N from the points of S_N. When two consecutive points are connected according to their flags, the face connecting them is implicitly declared feasible and efficient.

3.2 Better representation of the non-dominated set

As we have seen, D_{ex} stores efficient points that have been proved to be non-dominated. Assume that the branch and bound procedure is correct, that is D_{ex} contains exactly the extreme efficient points after the termination of the algorithm. We therefore obtain a set of points which may be non-connected as showed on Figure 3. In this case, although they are non-dominated, the points of D_{ex} are not sufficient to describe Y_N as for MOLP problems because no face of Y_N link two of these points so if a decision maker is not satisfied by any of the four solutions proposed, there is no way to choose an other efficient solution.

In S_N, some intermediate points are stored in addition to the extreme ones as shown in Figure 4. Compared to D_{ex}, S_N does not contain three but five more points because I_1 and I_2 are double points. In fact for I_1 we need one point to end the face starting in A and another one to start the face ending in B and it is the same for I_2 connecting B and C. Now, if the decision maker is not satisfied by any of the four extreme solutions, it is possible to compute quickly and propose him other efficient solutions more likely to be suitable for him.

3.3 Non extreme efficient points

When a new set of partially efficient points E is generated from a final node, we have seen that merging E and S_N is not sufficient. The update of S_N must

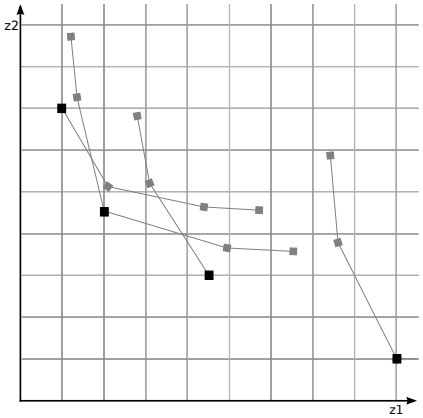

Fig. 3. Representation of D_{ex} with extreme points.

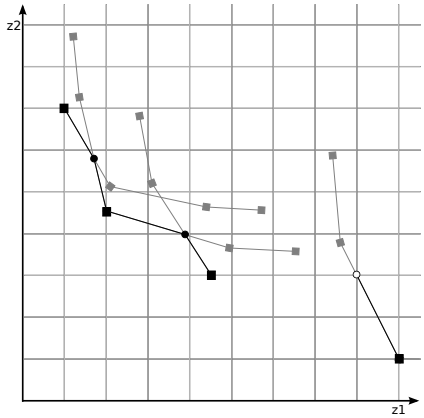

Fig. 4. Representation of S_N with extreme and guide points.

indeed be subtle enough to also add the intermediate points that will described exactly the current set of partially efficient points.

Because many situations can happen during the update of S_N, we will here focus on the description of the cases that are the most likely to appear. The update procedure considers one point at a time and is basically divided in two parts. The first one only checks if the current point dominates or is dominated by any point represented by S_N. The second part deals with the edge that may start at the current point. At each step, we assume that everything has been done correctly before so we only have to consider the current point and what is on its "right".
During all the solution, a special flag *nondom* is set on *true* or *false* depending on how were S_N and E just before the current point. *nondom* is set on *true* when E dominates S_N, otherwise its value is *false*. This flag is for example used to know when it is appropriate to add a intermediate points to shorten an edge of S_N.

Let x be the point of E currently tested for dominance. First the position of x compared with S_N has to be checked. If $c^1 x < c^1 s_{first}$ where s_{first} is the first point of S_N (i.e. x is on the left of S_N) then x is obviously added to S_N.

If x is not on the left of S_N, let s_1 be the last point such that $c^1 s_1 < c^1 x$. x will not be added if it is dominated by s_1.
As intermediate points matter, when s_1 has as successor s_2 and is connected to it, we need to check if a convex combination of them dominates x. To do so, we compute the projection x_{proj} of x onto the edge $[s_1, s_2]$ along the second objective's axis. If x is dominated by x_{proj} (Figure 5) then it is discarded, otherwise (Figure 6) it will be added to S_N. If the flag *nondom* is equal to *false*, x_{proj} is added to end the non-dominated edge starting in s_1 and *nondom* is set on *true*.

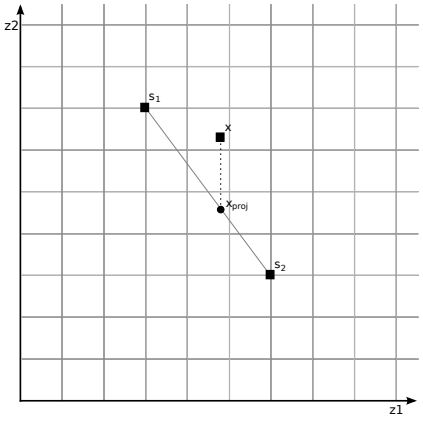

Fig. 5. Candidate point dominated by its projection onto an efficient edge.

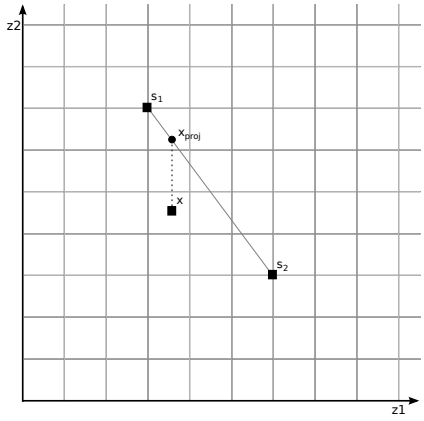

Fig. 6. Candidate point non-dominated by its projection onto an efficient edge.

In the case where x has been found to be non-dominated, it is possible that one or more points of S_N are dominated by it. All extreme points dominated by x are consequently removed. How intermediate points are treated will be explained later.
These first steps are summarized in Algorithm 1.

Now we know whether x has been added or not we have to consider the case where it has a successor x_2 in E. Since E is the non-dominated set of a MOLP problem, x and x_2 are connected if x_2 exists. We have to compare combinations of x and x_2 with each couple of consecutive points s_1 and s_2 such that $c^1x <= c^1s_2$ and $c^1s_1 <= c^1x_2$, that is we exclude couples of points ending before x or starting after x_2. For a given couple s_1 and s_2, we have two main possibilities.
First, assume that s_1 and s_2 are not connected. We can then focus only on s_1 because s_2 will be considered when we will shift to the next couple of point (s_2 will become s_1). If s_1 is dominated by a convex combination of x and x_2 then we simply remove it. Otherwise we have to remove the convex combinations dominated by s_1 to the edge that will be added to S_N, that is new intermediate points will be added if necessary.
Suppose now that s_1 is connected to s_2. Edges $[s_1, s_2]$ and $[x, x_2]$ can intersect in p. If p is found to be at the same time a convex combination of x and x_2 and of s_1 and s_2, the edges actually intersect in p. In this case (see Figure 7), p is added twice as the end of a face and the beginning of the other, depending on the value of the flag *nondom*. In Figure 7, p is added as the end of $[x, p]$ and then as the beginning of $[p, s_2]$ and also contains the corresponding variables values. An other possibility is that $[s_1, s_2]$ may overlap a part of $[x, x_2]$ like in Figure 8.

Algorithm 1 Update1 procedure

Require: S_N and x, a partially non-dominated point of the current node
determine s_1 and s_2 the two consecutive points of S_N such that $c^1 s_1 < c^1 e \leq c^1 s_2$
if s_1 does not exist **then**
Add x to S_N
else
if e is not dominated by d_1 **then**
if s_1 and s_2 are connected **then**
Compute the projection x_{proj} of x onto $[s_1, s_2]$ along the axis of the second objective
if x is not dominated by x_{proj} **then**
if *nondom* is false **then**
Add x_{proj} as the end of the edge starting in s_1
$non - dom \leftarrow true$
end if
Add x
end if
else
if x is non-dominated by s_2 **then**
Add x do S_N
$non - dom \leftarrow true$
end if
end if
Remove all the points explicitly belonging to S_N dominated by x
end if
end if

This case happens when *nomdom* is set on *true* (i.e. just before x, S_N was not dominating E) and s_1 dominates its projection onto $[x, x_2]$.

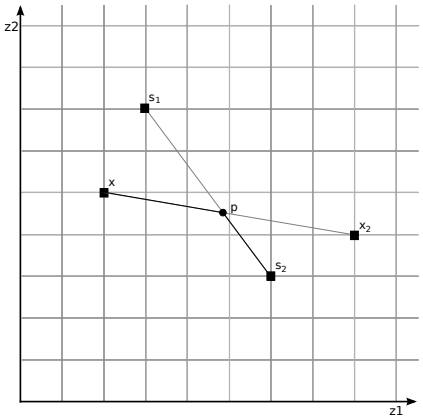

Fig. 7. Edges $[s_1, s_2]$ and $[x, x_2]$ intersect in p.

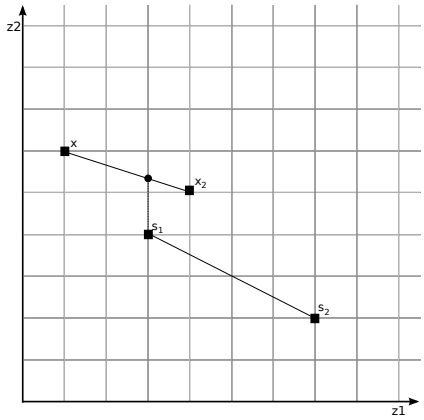

Fig. 8. $[s_1, s_2]$ overlap the end of $[x, x_2]$.

If x has no successor (i.e. it is the last point found in the current final node) and if an edge $[s_1, s_2]$ as previously defined exists, we just need to check if the end of the edge is dominated by x. The test is based on the same idea of projection but along the axis of the first objective function. In the example shown in Figure 9, x_{proj} will be added to start the remaining edge $[x_{proj}, s_2]$.

3.4 Finding a better lower bound set

In the original algorithm the fathoming test consists in comparing the ideal vector of the current node with D_{ex}. If the vector is dominated, then the node can obviously be fathomed and its successors will not be explored. The old definition of D_{ex} was inexact so nodes that had to be fathomed could be missed if the ideal vector was not dominated an extreme point but by and intermediate one. The updated version S_N is consequently a better lower bound set since it will allow to fathom more nodes.

Using the ideal vector in the fathoming test has the advantage to be very cheap. Computing it consists in fact in solving two single objective linear programing problems and then the test procedure only has to compare this single point with S_N. However inefficient nodes could be kept due to the distance between the ideal vector and the non-dominated points. The ideal vector is indeed "far away" from the non-dominated set due to the large number of non-dominated points and the conflicting nature of the objective functions. As a result, a large part of the combinatorial tree may be explored in vain because

Algorithm 2 Update2 procedure

Require: S_N and x, a partially non-dominated point of the current node
  if x has a successor in E **then**
    let x_2 be the successor of x {x and x_2 are connected}
    determine s_1 and s_2 the two consecutive points of S_N such that $c^1 s_1 < c^1 e \leq c^1 s_2$
    while $c^1 s_1 \leq c^1 x_2$ and s_2 exists **do**
      if s_1 and s_2 are connected **then**
        if $[s_1, s_2]$ and $[x, x_2]$ intersect in point p **then**
          if *nondom* is false (i.e. $[s_1, s_2]$ dominates $[x, x_2]$ before c) **then**
            Add c as the end of $[s_1, c]$ and as the beginning of $[c, x_2]$
          else
            Add c as the end of $[x, c]$ and as the beginning of $[c, s_2]$
          end if
          $nondom \leftarrow \neg nondom$
        else
          if a point x_{dom} of $[x, x_2]$ is dominated by $[s_1, s_2]$ and *nondom* is true **then**
            Add the intermediate point x_{dom} to S_N to end the segment $[x, x_{dom}]$
            $nondom \leftarrow false$
          end if
        end if
      else
        if s_1 is dominated by a point of the edge $[x, x_2]$ **then**
          remove s_1
          $nondom \leftarrow true$
        else
          if *nondom* is true **then**
            Add the projection s_{proj} of s_1 along c^2 onto $[x, x_2]$ to end the segment $[x, s_{proj}]$
          end if
          $nondom \leftarrow false$
          if $c^2 s_1 > c^2 x_2$ **then**
            Add the projection s'_{proj} of s_1 along c^1 onto $[x, x_2]$ to start the segment $[s'_{proj}, x_2]$
          end if
        end if
      end if
      if s_2 is dominated by a convex combination of x and x_2 **then**
        Remove s_2 from S_N
      end if
      $s_1 \leftarrow s_2$
      $s_2 \leftarrow$ the next point in S_N
    end while
  else
    if x dominates a part of $[s_1, s_2]$ and $c^2 x > c^2 s_2$ **then**
      Add the projection x_{proj} of x along c^1 onto $[s_1, s_2]$ to start the segment $[x_{proj}, s_2]$
    end if
  end if

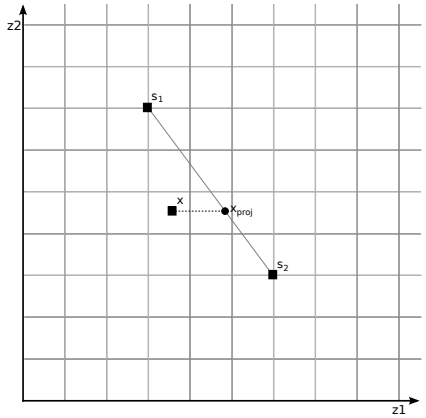

Fig. 9. Candidate point non-dominated by its projection onto an efficient edge.

Algorithm 3 Main update procedure

Require: S_N and E
  $nondom \leftarrow false$
  for all $x \in E$ **do**
    Update1(x, S_N)
    Update2(x, S_N)
  end for

the partially efficient points obtained are always dominated by S_N but their ideal vector is good enough to the nodes be considered as promising. Since the quality of a branch and bound relies strongly on the quality of its bounds, it could be preferable to use a more expensive but more accurate test.

The way to perform an exact fathoming test could be to compute for each node the corresponding efficient set E and compare it to S_N. If every points of E are dominated by S_N then the node can be fathomed, if not, the node is still promising and may lead to efficient solutions. Although this test is very accurate, it is also very expensive because all the points E have to be computed and compared to the points of S_N for each explored node. Thus, the computational effort needed by this test may compensate the gain of exploring less nodes.

There exists a compromise solution which will always outperform the complete computation and comparison. Actually, if we know the non-dominated set of a node, it is sufficient to scan it until a non dominated point is found. There is indeed no need to scan E entirely is we already know that the node will not be fathomed. For more literature about bound sets, see [8] and [15].

Moreover, we could compare the pseudo efficient points as soon as they are obtained until a non dominated point is generated. The whole non-dominated set of a node will be computed and compared to S_N only if it is dominated. In

this case, the node is fathomed so the expensive test has not been performed in vain.

Algorithm 4 Fathoming procedure

Require: S_N and the pseudo efficient set E of the current intermediate node
  Determine I_E the ideal vector of E
  if I_E is dominated by S_N **then**
    Fathom the current node
  else
    Seek at the beginning of the list L
    repeat
      Scan L forward
    until A non dominated point x_N is found or all the points of L have been scanned
    if L does not contain non dominated points **then**
      Fathom the current node
    else
      The node cannot be fathomed, branch on the next free binary variable
    end if
  end if

3.5 Tightening the search area

We have seen above that a good compromise can be used for the fathoming between the ideal vector and the whole efficient set, but the main problem remains the cost of this method. In fact, if the first non-dominated point is "far" in the list of pseudo efficient points, the algorithm will have to scan the list until it finds it and conclude that the node cannot be fathomed. Worse, the same will happen with the successors because the problem represented by the current node is a relaxed version of the children nodes problems where pseudo efficient solutions are at best as good as in the current node.

To avoid this problem we can simply add constraints to the original problem to bound the objective functions when we know where to search for efficient solutions. The procedure is described in Algorithm 5. On Figure 10, S_N crosses $Y_N(P)$, the non-dominated set of the problem P in the current node in point I. As P is a partial linear relaxation of the problems that the children nodes will generate, $Y_N(P)$ at least weakly dominates the non-dominated sets obtained in the final nodes of the branch. In other words, none of the final nodes of the current branch will provide solutions that dominate $Y_N(P)$. Consequently we can restrict the search area by forbidding the area where $Y_N(P)$ is already dominated by S_N. Since the fathoming test presented before stops as soon as a non-dominated point is found, we propose to only contract the search area and not to split it when dominated points of $Y_N(P)$ occur somewhere else than is its ends.

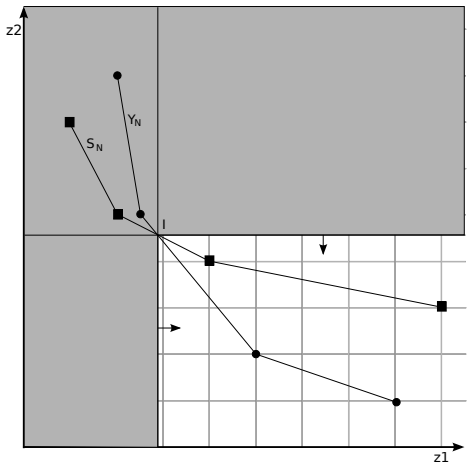

Fig. 10. New constraints to cut areas where the current branch cannot generate efficient solutions.

Now, when a expensive computation is done to try to fathom a node, this effort is not lost because it will improve the bounds on the objective functions and fasten the process with the successor nodes.

3.6 Branching strategy

In order to speed up the fathoming in the combinatorial tree, a good branching strategy has to be used. However, it is difficult to give an order to the variables to be fixed in the branch and bound algorithm in a general context because we don not know the constraint structure a priori.

Suppose that the scheme described in Section 3.5 is used during the process. In each node of the tree, a sub problem is solved, the pseudo efficient solutions are compared to D_{ex} and the bounds on the objective functions are tightened thanks to the first non dominated point found. The goal here is to fix the binary variables to reduce as soon as possible the search area. To do this, the branching could be done on the first relaxed binary variable entering or leaving the base in the first non dominated solution of the MOLP problem.

On Figure 11, the efficient set of the current intermediate node is compared to S_N and they intersect is I. The two extreme efficient points of the current node preceding and following I are A and B. If the solutions corresponding to A and B differ by a pivot involving a binary variable, it could be worth to branch directly on this variable. Thus we will consider two child nodes. The first successor node will generate a locally non-dominated set including A and the same with B for the second node. These sets are hoped to now intersect S_N respectively in I_A and I_B and consequently strengthen the constraints on the

Algorithm 5 Improved fathoming procedure

Require: S_N and the pseudo efficient set E of the current intermediate node
  Determine I_E the ideal vector of E
  if I_E is dominated by S_N **then**
    Fathom the current node
  else
    Sort in a list L the pseudo efficient points according to their z_1 value
    Seek at the beginning of the list L
    repeat
      Scan L forward
    until A non dominated point x_N is found or all the points of L have been scanned
    if L does not contain non dominated points **then**
      Fathom the current node
    else
      if x_N is connected to its predecessor **then**
        Let x_{left} be the last dominated convex combination of x_N and its predecessor
      else
        Let x_{left} be the predecessor of x_N
      end if
      Seek at the end of L
      repeat
        Scan L backward
      until A non dominated point x'_N is found
      if x'_N is connected to its successor **then**
        Let x_{right} be the last dominated convex combination of x'_N and its successor
      else
        Let x_{right} be the successor of x'_N
      end if
      On each direct successor of the current node, add or update the constraints on the objective functions:
      $c^1 x \geq c^1 x_{left}$
      $c^1 x \leq c^1 x_{right}$
      $c^2 x \geq c^2 x_{right}$
      $c^2 x \leq c^2 x_{left}$
    end if
  end if

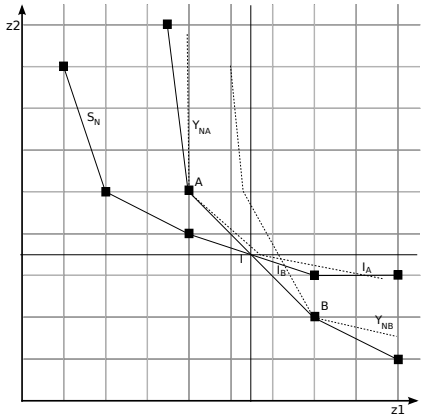

Fig. 11. Sub problems generated by good branchings may improve the constraints on the objectives.

objective functions more than I does. We can expect that this branching will lead earlier to efficient final nodes or fathom earlier non interesting nodes.

3.7 Initialization of the branch and bound

In order to fathom nodes in the branch and bound tree, it is necessary to know some "good" feasible solutions. However, a direct application of a branch and bound algorithm starts with no feasible solution. Consequently, no fathoming test can be applied in the first iterations of the algorithm, and the discovery of "good" feasible solutions may happen very late.

In a specific case, providing a set of good feasible solutions would be the role of an heuristic algorithm. But we work currently in a general context.

As we consider instances of mixed-integer linear programming that are rather small in a single-objective viewpoint, an exact solution of problems defined by a weighted sum scalarization should be done in reasonable time. Thus, we could compute "some" supported efficient solutions to initialize the algorithm. Indeed, if to solve one single-objective problem in reasonable time is realistic, the exact solution of a large number of single-objective problems may not be negligible. Thus, there are two possible choices : we use the classical dichotomous scheme to obtain the set of all extreme supported non-dominated points (the optimistic initialization), another possibility is to truncate the dichotomous scheme by avoiding the exact solution of a weighted sum problem if both points defining it are "near" (to be defined by experiments) to obtain well dispersed points. Some experiments will be necessary to observe the practical difficulty of these single-objective problems and to choose if the dichotomous scheme should be executed completely or not.

Next to this initialization, we obtain not only a set of good feasible solutions but a set of supported efficient solutions. This is a major difference with an initialization with an heuristic, which gives no information about the quality of the obtained set. In a branch and bound algorithm, we use a set of potentially efficient solutions we update during the enumeration. In particular, any solution of this set could be deleted if a new better solution is found. This cannot happen with the initial solutions that are proven to be efficient for the initial problem. This information must be used as far as possible. We can for example use the fact that all these solutions are supported and that we know the weight used to find these solutions, to define cuts in objective space. These cuts are usable whatever the used initialization. Adding cuts here does implies the same side-effects as for a combinatorial optimization problem, indeed we consider only LP relaxations in the node of the branch and bound tree, thus we solve MOLP problems in each steps.

If we execute the full dichotomy, we obtain in particular the usual lower bound set. Thus, we can use a set of cuts that restricts the search area to $(\operatorname{conv} Y_N)_N + \mathbb{R}^p_{\geqq}$. Moreover, as we restrict our study to the bi-objective case, the knowledge of lexicographic optimal points (whatever the initialization) allows us to find the ideal and nadir points. The restricted search area is therefore given by Figure 12.

If we cannot execute the full dichotomy, we can still use the same kind of cuts, knowing the ideal and nadir points, and the weights used to obtain the known supported efficient solutions (see Figure 13). In a more general configuration than Figure 13, we should use the fact that we know the local ideal point between two supported efficient points, if we have not considered the weighted sum problem these points define.

3.8 From a global to a local branch and bound

Consecutively to these cuts, if we "delete" the area dominated by the known supported non-dominated points, we can see that the area we need to explore can be clearly partitioned. This leads to another idea, rather than to apply a global multi-objective branch and bound algorithm (i.e. an algorithm that computes a complete set of efficient solutions), it could be interesting to execute several local branch and bound algorithms in each partition of the search area by adding the cuts restricting the search to each partition. Figures 14 and 15 illustrate these partitions.

The advantages of several local explorations rather than one global exploration are as usual, more nodes will be fathomed by infeasibility or dominance, and better feasible solutions will be found faster (in a hoped virtuous cycle). The drawback comes from possible redundancy of the obtained points.

This modification of the algorithm into a local branch and bound lead us to consider the two phase method (see [17]). The first phase would be as usual the dichotomous scheme to get the supported solutions (or at least a good subset of

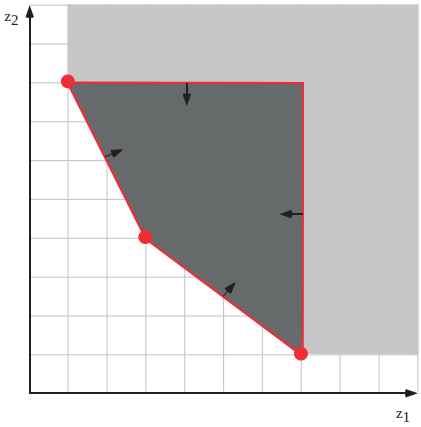

Fig. 12. Search area restricted by cuts in a case of the full execution of the dichotomy.

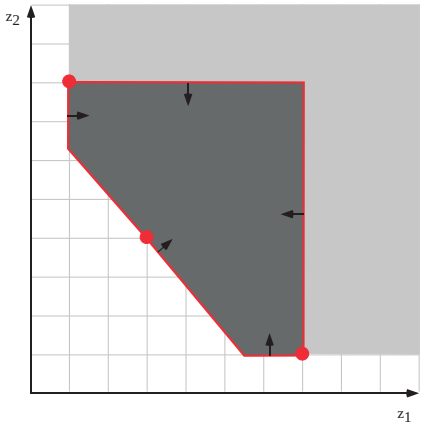

Fig. 13. Search area restricted by cuts in a case of a partial execution of the dichotomy.

them) and the second phase would be the branch and bound applied locally to each area. Experiments will provide a way to check if this strategy is interesting.

3.9 Exploration strategy

Initialization of a local branch and bound Suppose that we use the strategy of doing the exploration with several local search areas. We consider the two supported points delimiting the local area we are exploring. We know in particular the values of the binary variables for these two solutions and we can expect that most of the efficient solutions located in the local area will have a significant number of binary variables in common with the supported solutions defining this local area. Consequently, we can directly consider both MOLP problems with binary variables fixed with the same values as both supported solutions of the local area (these problems are leaves of the branch and bound tree). By solving the obtained MOLP problems, we will obtain new feasible solutions and we can hope to obtain "good" feasible solutions, to initialize the local branch and bound algorithm. The main difference with the global initialization (computing supported efficient solutions) is that there is no guarantee on the quality of the obtained solutions. Thus, these solutions will be considered as usual potentially efficient solutions.

Next to this initialization, the branch and bound algorithm (restricted to the local search area) can be applied as usually (from the root node with no fixed binary variable). Another possibility is to consider first a promising sub-tree of the branch and bound tree. This sub-tree is defined from the root node given by the last common node of both supported solutions defining the local search area, i.e. the node with all common binary variables fixed.

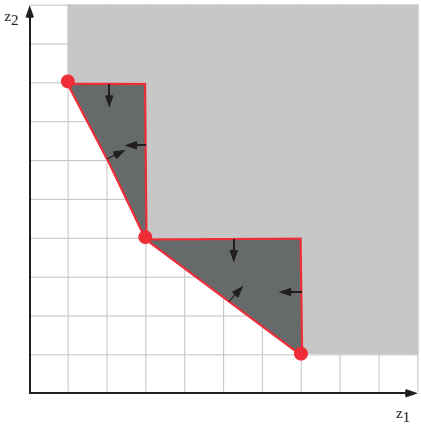

Fig. 14. Partitions of the search area restricted by cuts in a case of the full execution of the dichotomy.

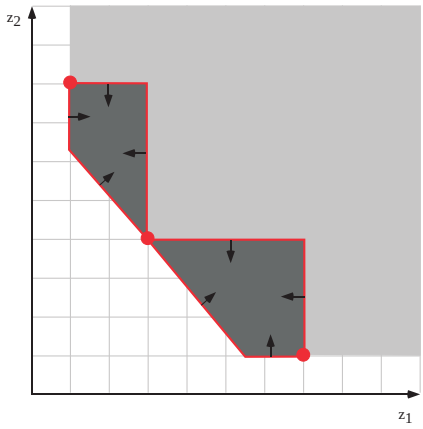

Fig. 15. Partitions of the search area restricted by cuts in a case of a partial execution of the dichotomy.

After this double initialization, the branch and bound algorithm can start from the root node (with no fixed variable) with a hoped accelerated execution, thanks to a set of very good feasible solutions.

4 Computational results

Some computational results are reported in this section from the implementation of some of the improvements presented in this paper. The algorithm used is the original branch and bound with S_N instead of D_{ex}. It is coded in C++ and runs under a Core2Duo 1.8GHz machine.
The algorithm was tested in some randomly generated mixed 0-1 MOLP problems of various size. The coefficients were randomly generated within the following intervals:

- $[-10, 10]$ for the objective coefficients of the continuous variables.
- $[-200, 200]$ for the objective coefficients of the binary variables.
- $[-100, -50]$ for the right hand sides.
- $[-20, 1]$ for the constraints coefficients.

No limit has been set on the sum of the binary variables to keep problems longer to solve. In fact it would limit the possible combinations of binary values and reduce the combinatorial tree to explore. Our goal is not to create test problems with small tree but on the contrary to have potentially huge trees and show our methods skip a non negligible part of it. The sparsity on the constraints matrix is set to 100%.

The results obtained from the tested problems are shown in Table 4. The parameters defined in the first column are: m the number of constraints, n_L the

number of linear variables and n_B the number of binary variables. The value of each parameter represents the average calculated from ten problems per problem type.
The times presented are really low compared to those presented in [13] and especially [12]. This is mainly due to the differences between the computers used for the experiments.

Table 1. Results

$m \times n_L \times n_B$	Number of Nodes	Final nodes	Efficient points	Extreme efficient points	Time (ms)	Time per node (ms)
10× 5× 5	30	7	8	7	37	1.2
20× 15× 5	27	7	13	13	140	5.2
20× 10× 10	90	12	14	13	1327	14.7
20× 5× 15	723	92	15	14	1243	1.7
30× 10× 10	174	36	13	10	3253	18.7

The three classes $20 \times 15 \times 5$, $20 \times 10 \times 10$ and $20 \times 5 \times 15$ provide interesting results. In fact, although the three of them have the same number of equations and the same number of variables, their results greatly differ depending on the ratio of binary variables :

- $20 \times 15 \times 5$ is the class that requires the less time, mainly because of the low number of binary variables. For this class, the combinatorial tree has 63 nodes and 42.9% of them have been visited on average. This large ratio is balanced by the low cost of the solutions of the sup-problems.
- $20 \times 10 \times 10$ is the "longest to solve" class : 1327 are necessary to explore 90 nodes (4.4% of the 2047 possible). The ratio of visited nodes is a lot lower than for the previous class but the time elapsed per node is three times as much.
- $20 \times 5 \times 15$ requires by far the biggest number of nodes explorations, but has on the contrary the lowest ration of visited nodes (only 1.1% of the 65535 possible). Moreover, only 1.7ms has been spent on each node on average, which implies that the problems solved are rather simple compared to the problems of the previous classes.

The ratio of visited nodes can be explained by the size of the tree. Indeed, the larger is the tree, the greater the number of the possibly fathomed nodes is, especially is we try to fathom them as soon as possible. Consequently, early cuts in the tree will have less effect if the tree is small because the number of remaining nodes will necessarily be smaller than in a large tree.
About the solution time, the ratio of linear variables seems to have an influence

but the results of the class $20 \times 10 \times 10$ disallow to conclude that the lowest is the ratio of linear variables, the simplest are the problems to solve. The time per node for the class $30 \times 10 \times 10$ is in the same range as the one of $20 \times 10 \times 10$, which could rather indicate that "well balanced" problems are the most difficult to solve.

5 Conclusions

MOMILP problems are useful for a large number of situations. Multiple objective functions are indeed useful to take several conflicting goals into account while mixing binary and linear variables allows to formulate the parameters (continuous) and the structure (discrete) of a system. In particular, *Multiple Objective Mixed Binary Linear Programming* (mixed 0-1 MOLP) represents a natural approach to model problems where the system structure consists in questions which can be answered by *yes* or *no*, like location problems.

In this paper we presented Mavrotas and Diakoulaki's branch and bound for 0-1 MOMILP problems and proposed several corrections and improvements. We first corrected the way to represent the efficient solution set to keep track of every efficient solutions, extreme or not. That lead us to use the set S_N instead of D_{ex}. The main difference between them is that S_N stores intermediate (and sometimes dominated) points in addition to the extreme efficient points considered by Mavrotas and Diakoulaki. Because it is able to describe exactly the non-dominated set of a MOMILP problem, S_N is preferable, but it is more difficult to store and to update efficiently all the points. We proposed a solution in the case of $p = 2$ objectives.

This improvement gave us a better upper bound set and we developed some ideas to use this set as much as possible to add interesting bounds on the objective function and consequently to fathom dominated nodes earlier. We also considered branching strategy that may be promising although we deal here with general problems. Moreover, using the branch and bound procedure locally in the two phases method instead of as a global algorithm is promising because it may restrict the search areas where the branch and bound is used. Further experiments should be done to have more insights on these propositions and the MOMILP problems in general, especially with more than two objective functions.

References

1. H.P. Benson and E. Sun. Outcome space partition of the weight set in multiobjective linear programming. *J. of Optim. Theory and Appl.*, 105(1):17–36, 2000.
2. H.P. Benson and E. Sun. A weight set decomposition algorithm for finding all efficient extreme points in the outcome set of a multiple objective linear program. *Eur. J. of Oper. Res.*, 139:26–41, 2002.
3. G.R. Bitran. Linear multiple objective programs with zero–one variables. *Mathematical Programming*, 13(1):121–139, 1977.

4. G.R. Bitran. Theory and algorithms for linear multiple objective programs with zero–one variables. *Mathematical Programming*, 17(1):362–390, 1979.
5. J. Climaco, C. Ferreira, and ME Captivo. Multicriteria integer programming: An overview of the different algorithmic approaches. *Multicriteria Analysis*, pages 248–258, 1997.
6. RF Deckro and EP Winkofsky. Solving zero-one multiple objective programs through implicit enumeration. *European Journal of Operational Research*, 12(4), 1983.
7. M. Ehrgott. *Multicriteria optimization*. Springer, 2005.
8. M. Ehrgott and X. Gandibleux. Bound sets for biobjective combinatorial optimization problems. *Computers and Operations Research*, 34(9):2674–2694, 2007.
9. A. Geoffrion. Proper Efficiency and the Theory of Vector Maximization. 1967.
10. H. Isermann. Proper efficiency and the linear vector maximum problem. *Operations Research*, pages 189–191, 1974.
11. H. Isermann. The enumeration of the set of all efficient solutions for a linear multiple objective program. *Operational Research Quarterly*, pages 711–725, 1977.
12. G. Mavrotas and D. Diakoulaki. A branch and bound algorithm for mixed zero-one multiple objective linear programming. *European Journal of Operational Research*, 107(3):530–541, 1998.
13. G. Mavrotas and D. Diakoulaki. Multi-criteria branch and bound: A vector maximization algorithm for Mixed 0-1 Multiple Objective Linear Programming. *Applied Mathematics and Computation*, 171(1):53–71, 2005.
14. LM Rasmussen. Zero-one programming with multiple criteria. *European Journal of Operational Research*, 26(1):83–95, 1986.
15. F. Sourd and O. Spanjaard. A Multiobjective Branch-and-Bound Framework: Application to the Biobjective Spanning Tree Problem. *INFORMS Journal on Computing*, 20(3):472, 2008.
16. R.E. Steuer. Multi criteria optimization: theory, computation, and application, 1985.
17. EL Ulungu and J. Teghem. The two phases method: An efficient procedure to solve bi-objective combinatorial optimization problems. *Foundations of Computing and Decision Sciences*, 20(2):149–165, 1995.
18. B. Villarreal and M.H. Karwan. Multicriteria integer programming: A (hybrid) dynamic programming recursive approach. *Mathematical Programming*, 21(1):204–223, 1981.