Management Science

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

A Branch and Bound Algorithm for a Class of Biobjective Mixed Integer Programs

Thomas Stidsen, Kim Allan Andersen, Bernd Dammann

To cite this article:
Thomas Stidsen, Kim Allan Andersen, Bernd Dammann (2014) A Branch and Bound Algorithm for a Class of Biobjective Mixed Integer Programs. Management Science 60(4):1009-1032. http://dx.doi.org/10.1287/mnsc.2013.1802

MANAGEMENT SCIENCE
Vol. 60, No. 4, April 2014, pp. 1009–1032
ISSN 0025-1909 (print) | ISSN 1526-5501 (online)

http://dx.doi.org/10.1287/mnsc.2013.1802

A Branch and Bound Algorithm for a Class of Biobjective Mixed Integer Programs

Thomas Stidsen
DTU Management, Technical University of Denmark, 2800 Copenhagen, Denmark, thst@man.dtu.dk

Kim Allan Andersen
CORAL, Department of Economics and Business, Aarhus University, 8000 Aarhus C, Denmark, kia@asb.dk

Bernd Dammann
DTU Compute, Technical University of Denmark, 2800 Copenhagen, Denmark, beda@dtu.dk

Most real-world optimization problems are multiobjective by nature, involving noncomparable objectives. Many of these problems can be formulated in terms of a set of linear objective functions that should be simultaneously optimized over a class of linear constraints. Often there is the complicating factor that some of the variables are required to be integral. The resulting class of problems is named multiobjective mixed integer programming (MOMIP) problems. Solving these kinds of optimization problems exactly requires a method that can generate the whole set of nondominated points (the Pareto-optimal front). In this paper, we first give a survey of the newly developed branch and bound methods for solving MOMIP problems. After that, we propose a new branch and bound method for solving a subclass of MOMIP problems, where only two objectives are allowed, the integer variables are binary, and one of the two objectives has only integer variables. The proposed method is able to find the full set of nondominated points. It is tested on a large number of problem instances, from six different classes of MOMIP problems. The results reveal that the developed biobjective branch and bound method performs better on five of the six test problems, compared with a generic two-phase method. At this time, the two-phase method is the most preferred exact method for solving MOMIP problems with two criteria and binary variables.

Keywords: biobjective optimization; integer programming; branch and bound
History: Received November 2, 2011; accepted May 28, 2013, by Dimitris Bertsimas, optimization. Published online in *Articles in Advance* January 6, 2014.

1. Introduction

Many planning problems involve noncomparable objectives. For the decision maker, this type of problems often consist in finding trade-offs between, e.g., the price and quality of a product. When applying single-objective optimization methods to such planning problems, the conflicting and noncommensurate objectives are scalarised to a single objective that can then be handled by standard optimization approaches. If the decision maker wants to avoid mixing noncomparable objectives, multiobjective optimization problem methods must be applied.

A very important subclass of multiobjective optimization problems is the class of multiobjective mixed integer programming (MOMIP) models. MOMIP models can be considered to be an extension of mixed integer programming (MIP) models where multiple (linear) objective functions are allowed. MIP models have formed a cornerstone of operations research (OR) for decades and are supported by a large coherent mathematical theory and advanced software. The progress in mathematical polyhedral theory over the last three decades has enabled the development of specialized MIP solvers, like CPLEX, GUROBI, and X-PRESS MP. These solvers routinely solve MIP models that are theoretically hard and of impressive size, like the traveling salesman problem, the knapsack problem, the set partitioning problem, etc. Furthermore, there are now specialized computer modeling languages, such as AIMMS, AMPL, GAMS, Mosel, MPL, or OPL-Studio, for stating MIP models for different optimization packages. For MOMIP models, however, neither solvers nor modeling languages exist.

As we see it, one of the main problems of multiobjective optimization is that even though a well-developed theory exists, best presented in Ehrgott (2005), there are no generally available computer tools for formulating and solving MOMIP models. As stated in a recent paper by Mavrotas and Diakoulaki (2005, p. 55): "Nevertheless, two are the main obstacles for a wider application of vector maximization methods: The scarcity of related software and their ability to handle large-sized problems."

We completely agree with Mavrotas and Diakoulaki (2005). In our opinion, there is no doubt that the development of a general branch and bound algorithm for MOMIP problems can be a very helpful tool for solving real-life multiobjective mixed integer linear problems, which is exactly the case in the single-objective example.

The natural way of developing a MOMIP solver would be to extend the single-objective branch and bound method to handle multiple objectives. However, the excellent survey on multiobjective combinatorial optimization by Ehrgott and Gandibleux (2000) states that not many papers exist on branch and bound methods for multiobjective mixed integer programming problems. But lately there has been a growing interest in developing branch and bound methods for MOMIP problems, in particular for the case in which the integer variables are required to be binary; see, for instance, Mavrotas and Diakoulaki (1998), Mavrotas and Diakoulaki (2005), Bukchin and Masin (2004), Masin and Bukchin (2008), and Sourd and Spanjaard (2008). Unfortunately, none of these methods have been applied in a wider context, and there seems to be consensus that the two-phase method for solving MOMIP problems with two objectives is the preferred method. However, our implementation of the two-phase method, first presented by Ulungu (1993), iteratively employs a single-objective MIP solver as a black-box solver. For MOMIP problems with many points in the so-called Pareto-optimal front, this leads to prohibitive execution times. Furthermore, the two-phase method is hard to extend using modern polyhedral methods like cutting planes or Dantzig–Wolfe decomposition. Finally, it seems difficult to extend the two-phase method to more than two objectives. One of the problems has been to determine the set of extreme points. Recently, two papers have been published on that topic; see Przybylski et al. (2010a) and Özpeynirci and Köksalan (2010). The latter paper is the first one to do computational experiments to find all extreme nondominated points (which may not be all nondominated points; see §2) for MOMIP problems with up to four objectives. A generalization of the two-phase method to solve multiobjective integer programming problems with $k > 2$ objectives has been presented by Przybylski et al. (2010b). They applied their approach to the three-objective assignment problem. The two-phase method is described in §3.1.

The purpose of this paper is twofold: first, to give an overview of recently developed branch and bound methods for solving MOMIP problems, in particular the ones with binary variables, and second, to present a general branch and bound algorithm for a special class of MOMIP problems, namely, where only two objective functions are allowed, the integer variables are required to be binary, and the continuous variables occur in at most one of the two objectives. Even though this may seem quite restrictive, this class of MOMIP problems contains all biobjective combinatorial problems, such as the biobjective knapsack problem and the biobjective assignment problem. Also, facility-location problems are covered by this class. The developed branch and bound algorithm is tested on six classes of well-known biobjective problems and compared with a generic two-phase method. It turns out that the general branch and bound algorithm performs better than the generic two-phase method on five of the six classes of biobjective problems. For one class, namely, biobjective set-covering problems, the generic two-phase method performs the best.

This paper is organized as follows. In §2, we describe the general MOMIP problem and give some preliminaries. In §3, we give a presentation of the two-phase method, followed by a review of newly developed branch and bound approaches for solving MOMIP problems. In §4, we describe the subclass of MOMIP problems for which we have developed a branch and bound method, and in §5, we present the branch and bound algorithm. In §6, some experimental results are given, and we conclude this paper in §7. Six appendices are enclosed at the end of this paper describing the test problems.

2. Preliminaries

In this section, we first describe the general form of MOMIP. After that, we give some preliminaries.

Let $c^1, \ldots, c^k \in \mathcal{R}^n$ be n-dimensional vectors, and let $h^1, \ldots, h^k \in \mathcal{R}^p$ be p-dimensional vectors. Let $z(x, y) = (z_1(x, y), \ldots, z_k(x, y)) = (c^1x + h^1y, \ldots, c^kx + h^ky)$ be k linear objective functions with nonnegative n-dimensional integer variables $x \in \mathcal{Z}^n_+$ and nonnegative p-dimensional continuous variables $y \in \mathcal{R}^p_+$. Let A be an $m \times n$ matrix, G be an $m \times p$ matrix, and $b \in \mathcal{R}^m$.

The general MOMIP model with k objective functions, m constraints, and a mix of integer and continuous variables is given as follows:

$$\min\{z(x, y) \mid Ax + Gy \geq b,\ x \in \mathcal{Z}^n_+,\ y \in \mathcal{R}^p_+\}. \qquad (1)$$

Let $\mathcal{X}$ denote the set of feasible solutions to (1), that is, $\mathcal{X} = \{(x, y) \in \mathcal{Z}^n_+ \times \mathcal{R}^p_+ \mid Ax + Gy \geq b\}$; $\mathcal{X}$ is also referred to as the feasible set in *decision space*. Let $\mathcal{Z} = \{z \in \mathcal{R}^k \mid z_i = c^ix + h^iy,\ (x, y) \in \mathcal{X},\ i = 1, 2, \ldots, k\}$ denote the corresponding feasible set in *criterion space*.

The MOMIP model given by Equation (1) is a classical linear programming problem with a mix of integer and continuous variables, extended to include k linear objective functions instead of just one objective function.

We will deploy the Pareto concept of optimality, which is based on the following binary relation. Let $z^1, z^2 \in \mathcal{R}^k$. Then

$$z^1 \prec z^2 \quad \Leftrightarrow \quad z_i^1 \le z_i^2 \quad i = 1, 2, \ldots, k \text{ and } z^1 \ne z^2.$$

A point $z^2 \in \mathcal{R}^k$ is *dominated* by $z^1 \in \mathcal{R}^k$ if $z^1 \prec z^2$.

The set of *efficient solutions* $\mathcal{X}_E$ in decision space is defined as

$$\mathcal{X}_E = \{(x, y) \in \mathcal{X} \mid \nexists\, (\bar{x}, \bar{y}) \in \mathcal{X}\colon z(\bar{x}, \bar{y}) \prec z(x, y)\},$$

and the set of *nondominated points* $\mathcal{Z}_N$ in criterion space is given by

$$\mathcal{Z}_N = \{z \in \mathcal{R}^k \mid z = z(x, y),\ (x, y) \in \mathcal{X}_E\}.$$

The nondominated points in $\mathcal{Z}_N$ can be partitioned into *supported* and *unsupported* points. The supported ones can be further subdivided into *extreme* and *nonextreme*. To this aim, let us define the following set:

$$\mathcal{Z}^{\geq} = \text{conv}\{\mathcal{Z}_N \oplus \{z \in \mathcal{R}^k\colon z \ge 0\}\},$$

where $\oplus$ denotes the usual direct sum. A point $z \in \mathcal{Z}_N$ is a supported nondominated point if it is on the boundary of $\mathcal{Z}^{\geq}$; otherwise it is unsupported. A supported nondominated point z is extreme if it is an extreme point of $\mathcal{Z}^{\geq}$; otherwise it is nonextreme. The set of efficient solutions are sometimes referred to as the set of *Pareto-optimal solutions*, and the set of nondominated points are sometimes referred to as the *Pareto-optimal front*.

The ideal point is defined as

$$z^{\text{ideal}} = \Big(\min_{(x,y)\in\mathcal{X}} z_1(x, y), \ldots, \min_{(x,y)\in\mathcal{X}} z_k(x, y)\Big),$$

and the nadir point is defined as

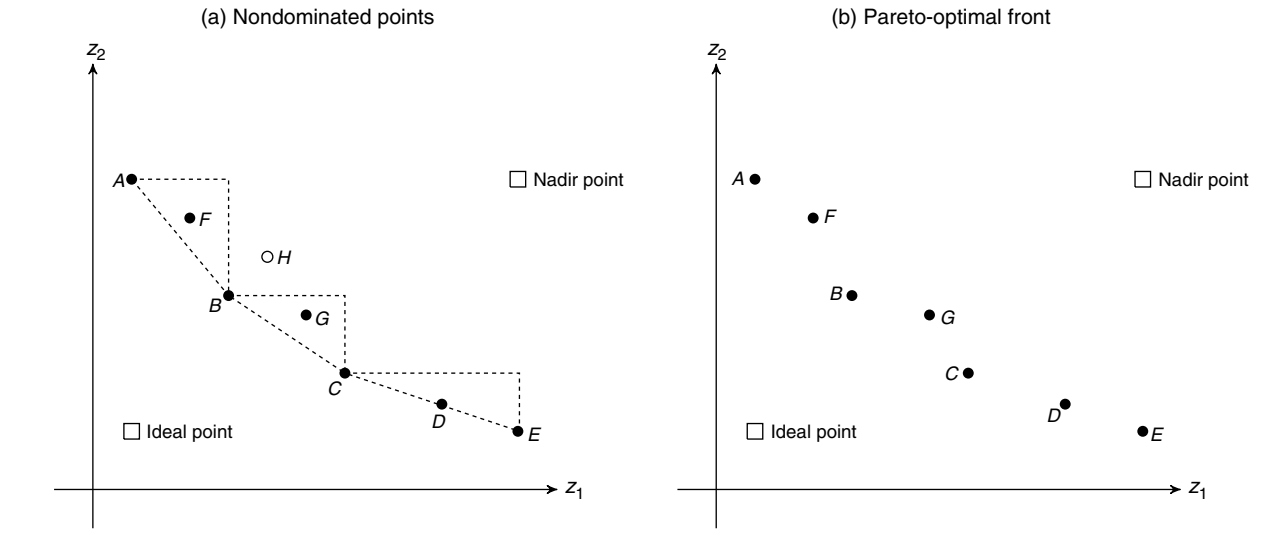
$$z^{\text{nadir}} = \Big(\max_{(z_1,\ldots,z_k)\in\mathcal{Z}_N} z_1, \ldots, \max_{(z_1,\ldots,z_k)\in\mathcal{Z}_N} z_k\Big).$$

The concepts are illustrated graphically in Figures 1(a) and 1(b) in the biobjective case ($k = 2$). Points A, B, C, and E are nondominated supported extreme points. Point D is a nondominated supported nonextreme point. Points F and G are nondominated unsupported points, and point H is a dominated point. In particular, and only possible in the biobjective case, the supported extreme nondominated points define a number of triangles in which unsupported nondominated points may be found. These triangles (shown with dotted lines) are illustrated in Figure 1(a), and the Pareto-optimal front is illustrated in Figure 1(b). The ideal point and the nadir point are illustrated with open squares.

3. Previous Approaches

Multiobjective mixed integer programming has been researched for more than 30 years. In the first part of this section, §3.1, we explain the two-phase method (see Ulungu and Teghem 1995), which presently is the most successful biobjective optimization method with binary variables. However, it is also capable of solving certain biobjective problems with continuous variables as well. In the following section, §3.2, we review the branch and bound methods developed for multiobjective optimization published in the last decade, when most of the promising branch and bound methods were published. In particular, we restrict ourselves to the case where the integer variables are binary. This is by far the most treated case in the literature to this date, covering all combinatorial problems. Furthermore, the branch and bound method

Figure 1 Illustration of Concepts



proposed in this paper requires that all integer variables are binary. In §6, we compare the branch and bound method developed in this paper with the two-phase method.

3.1. The Two-Phase Method

The two-phase method is a general approach for solving biobjective combinatorial problems. As the name suggests, the two-phase method divides the search for nondominated points into two phases. In Phase 1, the supported extreme nondominated points are found, and in Phase 2 the supported nonextreme and unsupported nondominated points are found.

We initialize Phase 1 of the two-phase method by finding the *upper left* (z^{UL}) and the *lower right* (z^{LR}) points. The upper left point can be found by solving the lexicographic optimization problem $\text{lexmin}\{z_1(x,y), z_2(x,y) \mid (x,y) \in \mathscr{X}\}$. The lower right point can be found by solving the lexicographic optimization problem $\text{lexmin}\{z_2(x,y), z_1(x,y) \mid (x,y) \in \mathscr{X}\}$.

In Phase 1, the set of supported extreme nondominated points are usually found using a parametric optimization problem. To define this problem, two supported extreme nondominated points, $z^+ = (z_1^+, z_2^+)$ and $z^- = (z_1^-, z_2^-)$, with $z_1^+ < z_1^-$, are needed. When using the two-phase method, two such points will always be available (at start, we use $z^+ = z^{\mathrm{UL}}$ and $z^- = z^{\mathrm{LR}}$). The parametric function $z_\lambda(x,y)$ is defined as follows:

$$\begin{aligned} \min \quad & z_\lambda(x,y) = \lambda z_1(x,y) + z_2(x,y) \\ \text{s.t.} \quad & (x,y) \in \mathscr{X}, \end{aligned} \tag{2}$$

where $\lambda \in \mathbb{R}_+$ is defined by the slope of the line connecting z^+ and z^-:

$$\lambda = \lambda(z^+, z^-) := (z_2^+ - z_2^-)/(z_1^- - z_1^+). \tag{3}$$

Let (x^*, y^*) denote an optimal solution of (2) for a given value of λ. It is well known (see Steuer 1986), that $(z_1(x^*,y^*), z_2(x^*,y^*))$ is a supported nondominated point.

The pseudocode for Phase 1 is shown in Figure 2. It is an NISE (noninferior set estimation) algorithm (Cohon 1978), and the idea was first applied to the biobjective transportation problem (Aneja and Nair 1979), where it was used to determine the set of supported extreme nondominated points.

The procedure first finds the upper left (z^{UL}) and the lower right (z^{LR}) points. These two points can be found by solving two lexicographic optimization problems as described above. The upper left point $z^{\mathrm{UL}} = (z_1^{\mathrm{UL}}, z_2^{\mathrm{UL}})$ is found by first minimizing z_1, adding a constraint that fixes the value of z_1 to its minimum value, and then minimizing z_2. To find the lower right point $z^{\mathrm{LR}} = (z_1^{\mathrm{LR}}, z_2^{\mathrm{LR}})$, we do the same

Figure 2 Phase 1—Finding Supported Extreme Nondominated Points

```
1  procedure Phase 1()
2     z^UL := (z1(x^UL, y^UL), z2(x^UL, y^UL)), where
        (x^UL, y^UL) is optimal for
        lexmin{z1(x, y), z2(x, y) | (x, y) ∈ X};
3     z^LR := (z1(x^LR, y^LR), z2(x^LR, x^LR)), where
        (x^LR, y^LR) is optimal for
        lexmin{z2(x, y), z1(x, y) | (x, y) ∈ X};
4     if (z^UL = z^LR) then STOP (only one
        nondominated point);
5     S := {z^UL, z^LR};
6     z+ := z^UL; z− := z^LR;
7     while (z+ ≠ z^LR) do
8        λ := λ(z+, z−);
9        solve (2) with optimal solution (x*, y*)
           and value z* = (z1(x*, y*), z2(x*, y*));
10       if (zλ(x*, y*) < λz1+ + z2+) then add z*
           between z+ and z− in S;
11       else z+ := z−;
12       if (z+ ≠ z^LR) then z− := Next(S, z+);
13    end while
14 end procedure
```

in the opposite order. In case the problem is infeasible or the two points z^{UL} and z^{LR} coincide, we are done. Given two supported extreme nondominated points z^+ and z^-, the *search direction* $\lambda = \lambda(z^+, z^-)$ defined by (3) is calculated, and (2) is solved using an MIP solver (such as CPLEX; IBM ILOG 2012). If the optimal solution (x^*, y^*) of (2) corresponds to a new supported extreme nondominated point, then the parametric objective function value $z_\lambda(x^*, y^*)$ must be less than the parametric objective function value of z^+ and z^- (line 10). The points z^+, z^*, and z^- then define two new search directions, and the **while** step is repeated on the points z^+ and z^* as well as on z^* and z^-. Otherwise, no new supported extreme nondominated point is found, and we proceed with the two next points in $\mathscr{S}$. Next$(\mathscr{S}, z)$ returns the point following z in $\mathscr{S}$. The set $\mathscr{S}$ is initialized with the two supported extreme nondominated points z^{UL} and z^{LR}. Whenever a new supported extreme nondominated point is found, the set $\mathscr{S}$ is updated. When no additional supported extreme nondominated points can be found, the set $\mathscr{S}$ contains the full set of supported extreme nondominated points, and the procedure stops.

Because supported nonextreme nondominated points or unsupported nondominated points may exist, it is not in general possible to find all nondominated points during the first phase. These points are found in Phase 2, which separately searches each triangle defined by the set of supported extreme nondominated points found in Phase 1; see Figure 1(a). The set of supported extreme nondominated points $\mathscr{S}$ found in Phase 1 may be ordered according to increasing values of the first objective, and a given triangle

is defined by two consecutive points in this list. This means that the first point in $\mathcal{S}$ is the point z^{UL}, the last point in $\mathcal{S}$ is the point z^{LR}, and the number of triangles to search in the second phase is one less the number of points in $\mathcal{S}$; see Figure 1(a).

Consider a particular triangle $\triangle(z^+, z^-)$ found in Phase 1 defined by the supported extreme nondominated points $z^+ = (z_1^+, z_2^+)$ and $z^- = (z_1^-, z_2^-)$, with $z_1^+ < z_1^-$. The local nadir point, the point in the triangle with the largest objective function value, is (z_1^-, z_2^+). The search for nondominated points in this triangle may be carried out in several ways and is often problem specific. One way is to find nonoptimal solutions to (4), where $\lambda \in \mathbb{R}_+$ is defined by (3):

$$\begin{aligned} \min \quad & z_\lambda(x, y) = \lambda z_1(x, y) + z_2(x, y) \\ \text{s.t.} \quad & z_1 \le z_1^-, \\ & z_2 \le z_2^+, \\ & z_\lambda(x, y) \le \mathrm{UB}, \\ & z_\lambda(x, y) \ge \mathrm{LB}, \\ & (x, y) \in \mathcal{X}, \end{aligned} \tag{4}$$

where UB and LB are, respectively, upper and lower bounds on the parametric objective function value.

To illustrate Phase 2, a general generic pseudocode is given in Figure 3. The procedure is called with the set of supported extreme nondominated points $\mathcal{S}$ found in Phase 1. In lines 3 and 4, the procedure is initialized. In line 3, the counter k is initialized to 1, and the current set of nondominated points $\bar{\mathcal{Z}}_N$ is initialized with the set $\mathcal{S}$. In line 4, the two points z^+ and z^- in $\mathcal{S}$, which define the first triangle, are initialized.

The **while** step, which begins in line 5 and finishes in line 12, searches the triangles one by one for nondominated points.

In line 6, lower and upper bounds on the parametric value of the objective function in a given triangle $\triangle(z^+, z^-)$ are initialized. The lower bound is attained at the two points z^+ and z^-, and the upper bound is attained at the local nadir point. In line 7, problem (4) is initialized (set up), and in line 8, two no-good constraints are added to the problem. Assume that $z^+ = (z_1^+(x^+, y^+), z_2^+(x^+, y^+))$. Then one of the two no-good constraints (see Hooker 2000), corresponding to z^+, is given by

$$\sum_{j=1}^{n} \{x_j \mid x_j^+ = 0\} + \sum_{j=1}^{n} \{1 - x_j \mid x_j^+ = 1\} \ge 1. \tag{5}$$

The no-good constraint (5) separates the solution (x^+, y^+) from the feasible set to (4). The other no-good constraint is similar to (5), but corresponds to the point z^-. In line 9, procedure Inside($\triangle(z^+, z^-)$) is called. It finds all nondominated points in the triangle $\triangle(z^+, z^-)$. When all triangles have been searched in the **while** step, the set $\bar{\mathcal{Z}}_N$ contains the full set of nondominated points, and the procedure stops.

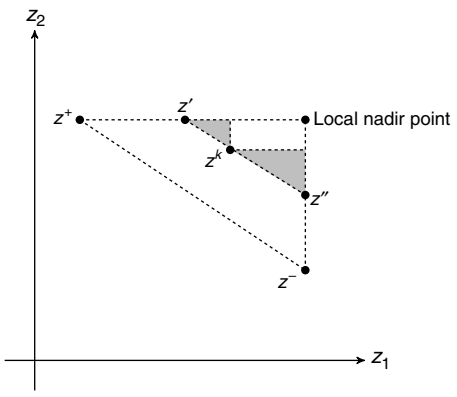
Figure 3 Phase 2—Finding Unsupported Nondominated Points

```
1  procedure Phase 2(S)
2      Comment: The first point in S is z^UL, and the
         last point in S is z^LR;
3      k := 1; Z̄_N := S;
4      z+ := z^UL; z− := Next(S, z+);
5      while (z+ ≠ z^LR) do
6          LB := λz1+ + z2+; UB := λz1− + z2+;
7          Initialize problem (4);
8          Add two no-good constraints to (4)
               corresponding to z+ and z−;
9          Inside(△(z+, z−));
10         z+ := z−;
11         if (z+ ≠ z^LR) then z− := Next(S, z+);
12     end while
13     Z_N := Z̄_N;
14 end procedure
```

```
1  procedure Inside(△(z+, z−))
2      while (LB ≤ UB) do
3          Solve (4). If (4) is infeasible return;
4          Let (x^k, y^k) be an optimal solution to (4)
               with value z^k = (z1(x^k, y^k), z2(x^k, y^k));
5          if (NonDom(z^k)) then Z̄_N := Z̄_N ∪ {z^k};
6          Add a no-good constraint to (4)
               corresponding to z^k;
7          LB := λz1^k + z2^k; k := k + 1;
8      end while
9  end procedure
```

Procedure Inside($\triangle(z^+, z^-)$) finds all nondominated points in $\triangle(z^+, z^-)$; see Figure 4. In each loop in the **while** step problem (4) is solved. If it is infeasible all nondominated points in the triangle $\triangle(z^+, z^-)$ have been found, and the procedure returns. Otherwise, an optimal solution (x^k, y^k) has been found. If the corresponding point z^k in criterion space is nondominated, it is added to the current set of nondominated points $\bar{\mathcal{Z}}_N$ in line 5. A no-good constraint similar to (5) that separates the optimal solution (x^k, y^k) from the feasible set to problem (4) is added in line 6, and in line 7, the lower bound is updated. Note that problem (4) is

Figure 4 Triangle Search



augmented with an extra constraint in each loop in the **while** step.

A few remarks on Figure 3:

• Sometimes it may be possible to obtain a better (lower) upper bound than the one given by the local nadir point. This is in general only possible if the problem contains integer variables and no continuous variables, and both objective functions have integer coefficients. A prominent example of this is the assignment problem, where the upper bound can be strengthened considerably; see, for instance, Tuyttens et al. (2000), Visée et al. (1998), Przybylski et al. (2008), and Pedersen et al. (2008).

• Any solver that can solve MIP problems can be used as function Solve( ). In this paper we used CPLEX MIP Solver version 12.1.1 (IBM ILOG 2012).

However, the procedure shown in Figure 3 is just one of several possible search strategies to explore the triangle $\triangle(z^+, z^-)$ for nondominated points. One of the most successful enumeration strategies, and presently the reference strategy, are ranking (K-best) methods. These algorithms rank points with increasing values of the parametric objective function. In particular, these methods are useful when there exists an efficient algorithm for the single-objective problem. Some examples are the assignment problem by Przybylski et al. (2008), the multimodal assignment problem by Pedersen et al. (2008), and the minimum cost network flow problem by Raith and Ehrgott (2009). The problems considered in these papers are binary, and the authors have been able to develop upper bounds (better than the bound corresponding to the local nadir point), which limits the search for nondominated points in the triangles considerably. More examples can be found in Przybylski et al. (2011).

3.2. Previous Research

The excellent survey on multiobjective combinatorial optimization by Ehrgott and Gandibleux (2000) states that at the time not many papers existed on branch and bound methods for multiobjective optimization. These papers are Klein and Hannan (1982), Kiziltan and Yucaoglu (1983), White (1984), and Visée et al. (1998). However, since the publication of Ehrgott and Gandibleux (2000), i.e., over the last decade, a few more branch and bound methods have been proposed. In this section, we give an overview of the newly developed branch and bound methods for finding the entire set of nondominated points to multiple objective linear programs (LPs) where some or all of the variables are required to be binary, and the remaining variables are continuous. The reason for this choice is that these branch and bound methods are the most successful until now. Furthermore, this allows us to draw some observations that we can use in the development of the branch and bound algorithm presented in this paper.

When the integer variables are required to be binary, the general MOMIP problem (1) reduces to the following problem:

$$\min\{z(x, y) \mid Ax + Gy \geq b,\ x \in \mathcal{B}^n,\ y \in \mathcal{R}^p_+\}, \qquad (6)$$

where $\mathcal{B}^n$ is the set of n-dimensional binary vectors.

A depth first search branch and bound method for problem (6) is presented in Mavrotas and Diakoulaki (1998). The method can find the set of nondominated points. Whenever a final node (all binary values have been set) is reached, the resulting multiobjective linear program (MOLP) problem is solved. The corresponding set of nondominated points are called partially nondominated points (because they need not be nondominated for the entire problem (6)) and are stored in the set D_{ex} (only the criterion vectors of the efficient extreme solutions are stored). The set D_{ex} is updated at each final node. (Criterion vectors that are dominated are removed, and nondominated points are added.) Fathoming a node is done if it is infeasible or if the ideal point of the node is dominated by some partially nondominated point stored in D_{ex}. A number of improving steps have been added in a second paper; see Mavrotas and Diakoulaki (2005). Because of the computational effort involved in solving a MOLP problem at each final node, the method is capable of solving only small and medium-sized problems.

Unfortunately, it turns out that there may be dominated points in the output of the algorithm. The problem is that it is not enough to store the criterion vectors for the extreme efficient points at each node. It may be necessary to store also pseudo-nondominated points (i.e., points dominated only by a convex combination of two other extreme partially nondominated points are kept in D_{ex}). This issue has been addressed in Vincent (2009) and Vincent et al. (2013) and corrected in the biobjective case. In particular, a number of lower bound sets have been introduced in Vincent et al. (2013), and encouraging preliminary experimental results for randomly generated problems with up to 60 variables (30 binary) and 60 constraints are reported.

The remaining papers reviewed in this section propose branch and bound methods for problem (6) in the case when there are no continuous variables.

The branch and bound framework developed in Sourd et al. (2006) and Sourd and Spanjaard (2008) is identical to the usual branch and bound procedure (with one objective) in the branching part, but differs in the bounding part, which is performed via a set of points rather than a single (ideal) point; see Mavrotas and Diakoulaki (1998, 2005). The current node N can be discarded if a hypersurface can

be identified that separates the set of feasible points in the subproblem (corresponding to node N) from the set of potentially nondominated points. A simple example of a hypersurface is a straight line (in two dimensions), but Sourd and Spanjaard (2008) uses a more intelligent hypersurface, which enables better bounding. This hypersurface is, however, problem dependent. Promising experimental results for the biobjective spanning tree problem are presented.

The procedure developed in Bukchin and Masin (2004) and Masin and Bukchin (2008) is similar to the usual branch and bound procedure. In the multiobjective branch and bound procedure, a surrogate objective function $\alpha(x)$ is used. Assume that $\hat{x}$ is a partial solution (corresponding to a node in the branch and bound tree). Let $\mathcal{X}_N$ be the current set of nondominated points (which may be initialized to a set of known points or to a dummy point if no feasible points are known) that is updated during the procedure. Also, let $\hat{z}_i(\hat{x})$ be a lower bound on the ith objective function value given the partial solution $\hat{x}$. Then the surrogate objective function is defined as $\hat{\alpha}(\hat{x}) = \max_{z^e \in \mathcal{X}_N}(\min_{1 \le i \le k}((\hat{z}_i(\hat{x}) - z_i^e)/\Delta_{ie}))$, where Δ_{ie} is a positive scaling coefficient. A node in the branch and bound tree is fathomed if $\hat{\alpha}(\hat{x}) \ge 0$. If this condition is satisfied, no solution further down the branch and bound tree can lead to a nondominated point. No advice for selecting which variable to branch on is given. The procedure is illustrated in a small scheduling example, but no computational results are reported.

In Mezmaz et al. (2007), a grid-based parallel approach where one host plays the role of a farmer and all the other hosts play the role of workers is suggested. The farmer hosts the coordinator, and each worker hosts as many branch and bound processes as it has processors. Each worker also maintains a local Pareto front. The description of the procedure is generic. It is tested on a large flow-shop problem, and the Pareto-optimal front is found.

Having gone through the two-phase method and the newly developed branch and bound methods for solving multiple-objective problems with a mix of binary and continuous variables, we can make the following observations:

- The most promising of the existing branch and bound methods seem to be the methods by Mavrotas and Diakoulaki (1998), Mavrotas and Diakoulaki (2005) (corrected by Vincent et al. 2013 in the biobjective case) and Sourd and Spanjaard (2008). The latter approach seems to be able to solve larger problems than earlier methods. However, it requires some work to define an appropriate hypersurface that can separate the set of feasible solutions in the subproblem from the set of potential nondominated points. Also the method suggested by Bukchin and Masin (2004) and Masin and Bukchin (2008) seems promising. However, at this time no experimental results have been reported for this procedure.
- It is important to be able to obtain a good lower bound of the subproblems (corresponding to nodes in the enumeration tree). This is usually achieved by calculating the ideal point of a subproblem (for each objective, calculate the best possible value that can be obtained in the subproblem). However, this may not be a good lower bound (a bound that can fathom many nodes in the enumeration tree). The papers by Bukchin and Masin (2004) and Masin and Bukchin (2008) propose a way to solve this problem by introducing the concept of separation between reachable solutions (from the current partial solution corresponding to a node in the enumeration tree) and solutions that are not dominated by the current upper bound of the best solutions found so far.
- In general, the proposed branch and bound methods have not had much success; they have mostly been used only by their authors.
- Overall, it seems that the two-phase method is the best procedure for solving biobjective binary programs. In particular, if good lower and upper bounds can be found, it may be essential for the performance of the method. This is the case in, for example, the biobjective assignment problem. We refer to Przybylski et al. (2008) for a discussion of this. The two-phase method has been used by a number of authors throughout the years.

4. Problem Classification

In this section, we present the simplified set of MOMIP models for which we will develop a branch and bound algorithm. Like most other branch and bound models developed earlier (see §3.2), we assume that all integer variables are binary.

We make the following assumptions:

1. We only consider biobjective models, i.e., $k = 2$.
2. The integer variables are binary.
3. We only allow continuous variables to be part of *one* of the two objectives.

This means that the problem which we are considering can be expressed as follows:

$$\min\{(c^1x, c^2x + h^2y) \mid Ax + Gy \ge b,\ x \in \mathcal{B}^n,\ y \in \mathcal{R}_+^p\}. \quad (7)$$

A special case of problem (7) occurs when there are no continuous variables. The resulting problem is the biobjective combinatorial optimization problem. One example is the biobjective assignment problem. That problem may have an exponential number of nondominated points, and it is known to be $\mathcal{NP}$-complete and $\sharp\mathcal{P}$-complete; see Ehrgott (2005, Chap. 9). Another example is the biobjective knapsack problem, which is $\mathcal{NP}$-hard; see Ehrgott (2005, Chap. 10). In particular, this implies that problem (7) also has all these characteristics.

4.1. Problem Modifications

Before applying the biobjective algorithm described in §5, we introduce some simple problem modifications. These modifications are not strictly necessary, but they simplify the algorithm.

We will only consider minimization problems, in both objectives. Hence, maximization objectives must be multiplied by -1. Given a multiobjective minimization problem, like the one given in Equation (7), we now proceed to scale the coefficients of the objective functions.

4.1.1. Normalization. Normalization of the MOMIP problem is achieved through lexicographic optimization as explained in §3.1. We first find the upper left and the lower right points z^{UL} and z^{LR}. In case the problem is infeasible or the two points z^{UL} and z^{LR} coincide, we are done. Otherwise, we know that the Pareto-optimal front will reside in the grey rectangle shown in Figure 5(a).

In Figure 5(a), the grey rectangle is spanned by the two corner points z^{UL} and z^{LR}, the ideal point $z^{\mathrm{ideal}} = (\min_{(x,y)\in\mathscr{X}} z_1(x,y), \min_{(x,y)\in\mathscr{X}} z_2(x,y))$, and the nadir point $z^{\mathrm{nadir}} = (\max_{(z_1,z_2)\in\mathscr{Z}_N} z_1, \max_{(z_1,z_2)\in\mathscr{Z}_N} z_2)$, where $\mathscr{X}$ is the set of feasible solutions to (7), and $\mathscr{Z}_N$ is the set of nondominated points to (7). Given the two points z^{UL} and Z^{LR}, the two points z^{ideal} and z^{nadir} are easily found. For convenience, we rescale the rectangle by normalizing the two objective functions $z_1(x,y)$ and $z_2(x,y)$ according to Equation (8):

$$\bar{z}_1(x,y) = \frac{c^1x - z_1^{\mathrm{UL}}}{z_1^{\mathrm{LR}} - z_1^{\mathrm{UL}}}, \quad \bar{z}_2(x,y) = \frac{c^2x + h^2y - z_2^{\mathrm{LR}}}{z_2^{\mathrm{UL}} - z_2^{\mathrm{LR}}}. \tag{8}$$

The effect of the rescaling is illustrated in Figure 5(b). It is important to understand that the set of efficient solutions $\mathscr{X}_E$ is unaffected by this transformation (but the nondominated points are rescaled and translated).

Finally, we introduce a weighted objective

$$\bar{z} = \bar{z}_1 + \bar{z}_2 = \frac{1}{z_1^{\mathrm{LR}} - z_1^{\mathrm{UL}}} c^1x + \frac{1}{z_2^{\mathrm{UL}} - z_2^{\mathrm{LR}}}(c^2x + h^2y) - \left(\frac{z_1^{\mathrm{UL}}}{z_1^{\mathrm{LR}} - z_1^{\mathrm{UL}}} + \frac{z_2^{\mathrm{LR}}}{z_2^{\mathrm{UL}} - z_2^{\mathrm{LR}}}\right).$$

The final version of the rescalarised problem is given as follows:

$$\begin{aligned} \min\ & \bar{z} \\ \text{s.t.}\ & Ax + Gy \geq b \\ & x \in \mathscr{B}^n,\ \ y \in \mathscr{R}_+^p. \end{aligned} \tag{9}$$

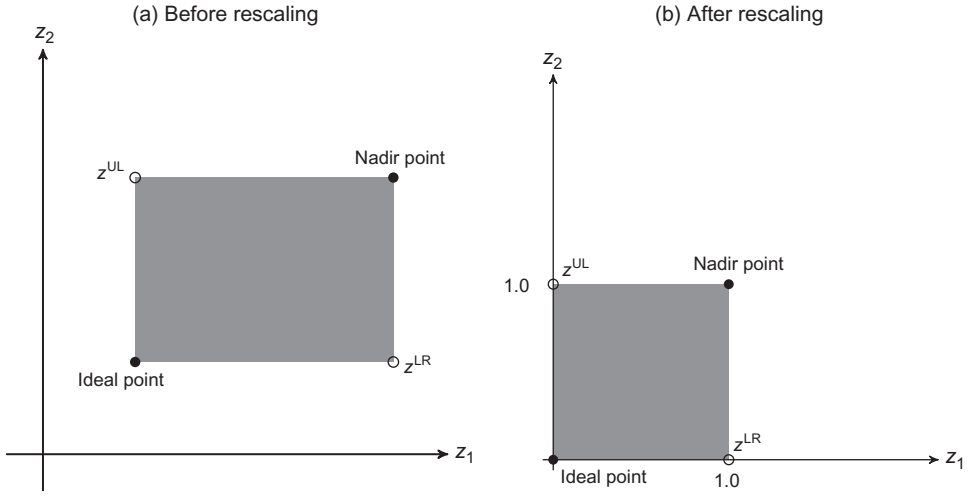
5. The Biobjective Branch and Bound Algorithm

The biobjective branch and bound (BOBB) algorithm resembles a standard branch and bound algorithm (Land and Doig 1960), but differs in two important aspects. Here we describe the most basic BOBB algorithm applied to the rescalarised problem (9).

The BOBB algorithm starts by solving a relaxed version of the problem (9), where all (binary) variables $x \in \mathscr{B}^n$ are relaxed to $x \in [0,1]^n$. Given the relaxed version of problem (9), we now have a single-objective LP problem that can be solved efficiently by a standard linear program solver. The BOBB algorithm will, just like the standard branch and bound algorithm, apply branching on the relaxed binary variables. During execution, an enumeration tree data structure will be kept and maintained. Initially, the fully relaxed problem is solved using the LP solver, arriving at the root solution in the enumeration tree. The optimal LP solution as well as some additional solution information are saved in the enumeration tree data structure.

Each node in the enumeration tree correspond to a subproblem of (9), where some of the binary variables

Figure 5 The Pareto-Optimal Front Resides in the Grey Area



have been fixed to zero or one. In each node a six-tuple of values $(\bar{z}_{\text{LP}}, \bar{x}_{\text{LP}}, F_0, F_1, F/IF, I)$ are saved. The six-tuple corresponds to the following:

1. $\bar{z}_{\text{LP}}$: The LP objective value after LP optimization.
2. $\bar{x}_{\text{LP}}$: The optimal solution to the LP optimization is $(\bar{x}_{\text{LP}}, \bar{y}_{\text{LP}})$. Only $\bar{x}_{\text{LP}}$ is part of the six-tuple.
3. F_0: The set of relaxed binary variables that has been fixed to 0.
4. F_1: The set of relaxed binary variables that has been fixed to 1.
5. F/IF: Is the LP feasible or infeasible?
6. I: This set keeps a record of all previously seen optimal binary solutions. If $\bar{x}_{\text{LP}}$ is binary it is added to I.

The last element in the six-tuple is new, compared to traditional branch and bound procedures. During execution it may happen that a feasible solution to (9) is found. In a traditional branch and bound algorithm, the corresponding node can be fathomed in such a situation. However, this is not the case in biobjective problems; see §5.3. The set I is a list of the binary part x of feasible solutions (x, y) to (9) that has occurred in the unique path from the root node to the current node in the enumeration tree.

We use the notation $OPT(-, -, F_0, F_1, -, I)$ in the application of an LP solver for the current values of F_0, F_1, and I. When the LP solver is applied to a subproblem specified by a six-tuple, we use the following notation: $(\bar{z}_{\text{LP}}, \bar{x}_{\text{LP}}, F_0, F_1, F/IF, I) \leftarrow OPT(-, -, F_0, F_1, -, I)$. The notation means that before applying the LP solver to problem (9), the binary variables given in F_0 and F_1 are fixed. Furthermore, for each previously seen optimal binary solution stored in I, we are adding a constraint; see (10) in §5.3. The LP solver finds the values $\bar{z}_{\text{LP}}$, $\bar{z}_{1,\text{LP}}$, $\bar{z}_{2,\text{LP}}$, $\bar{x}_{\text{LP}}$, and $\bar{y}_{\text{LP}}$, and furthermore sets F/IF to either F (feasible) or IF (infeasible).

For the root solution, the set of relaxed binary variables that has been fixed is naturally empty, i.e., $F_0 = \{\}$ and $F_1 = \{\}$. Furthermore, there are no previously seen optimal binary solutions, i.e., $I = \{\}$. When the root solution has been saved as the six-tuple, the algorithm enters the main loop.

In the main loop, first a relaxed solution, saved as a six-tuple in the set S of nodes that still need to be processed, is retrieved. If S is empty, the algorithm terminates. Given the six-tuple, the BOBB algorithm attempts to fathom the solution, like the classical branch and bound algorithm. Three kinds of fathoming are possible:

1. infeasibility fathoming,
2. bound fathoming,
3. integer fathoming (which is not really fathoming, but standard variable branching).

If none of these fathomings are possible, standard (binary) branching on a fractional variable is performed. In the subsections below, we describe each of the three fathoming cases.

5.1. Infeasibility Fathoming

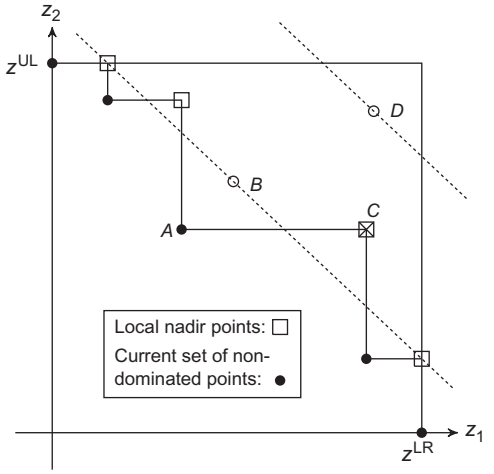
If the LP is infeasible, the current solution (node in the enumeration tree) can be fathomed. This is exactly as in the standard branch and bound algorithm.

5.2. Bound Fathoming

In contrast to the standard branch and bound algorithm, the incumbent is *not* one single solution, but the current set of nondominated points $\bar{\mathcal{X}}_N$. We first graphically show how bounding is possible, then we formalize the bounding procedure.

The bounding situation is depicted in Figure 6. Point $B = (b_1, b_2)$ with objective function value $b_1 + b_2$ cannot be fathomed because even though it is actually dominated by an existing solution $A = (a_1, a_2)$ in the current set of nondominated points $\bar{\mathcal{X}}_N$, it is possible that another later point, say, $v = (v_1, v_2)$ with $v_1 + v_2 > b_1 + b_2$, can actually be a part of the final set of nondominated points $\mathcal{X}_N$. The objective function value level curve of B is shown as the dashed line through the point. To be able to fathom a current point, the objective function value of the point has to be larger than the objective function value of any nondominated point of the current set of nondominated points. To evaluate this, we compare the objective function value of the current point in criterion space with the objective function values of the local nadir points. The four local nadir points in Figure 6 are marked with open squares around them. The *worst* local nadir point $C = (c_1, c_2)$ (the one with the largest objective function value) is further marked with a cross. If the objective value of the current point is larger than the objective value of all local nadir points (first suggested in Tuyttens et al. 2000), the point can be bounded and fathomed. In Figure 6, point B cannot be bounded, but point D can. Initially, the two extreme points of the normalized problem $(0, 1)$ and $(1, 0)$ constitute the current set of nondominated points $\bar{\mathcal{X}}_N$, and the largest objective function

Figure 6 Bounding



value of the local nadir points (i.e., the nadir point) is $WORST_{\text{local nadir}} = 2$.

Given the current set of nondominated points $\bar{\mathcal{Z}}_N$, we will now more formally describe the bounding procedure. Because of the initialization of the BOBB algorithm, i.e., the lexiographic optimization and normalization, when the BOBB algorithm enters the main loop, the current set of nondominated points $\bar{\mathcal{Z}}_N$ consists of two points $(0, 1)$ and $(1, 0)$. For any future point in the current set of nondominated points $z_i = (z_i^1, z_i^2) \in \bar{\mathcal{Z}}_N$, we know that $z_i^1 \in]0, 1[$ and $z_i^2 \in]0, 1[$. We assume that the points are ordered according to increasing values of the first objective. For each point z_i and the following point z_{i+1} in the current set of nondominated points $\bar{\mathcal{Z}}_N$, we know that $z_i^1 < z_{i+1}^1$ and $z_i^2 > z_{i+1}^2$.

Given the current set of nondominated points $\bar{\mathcal{Z}}_N$, we define a set of local nadir points between each pair of points z_i and z_{i+1} in $\bar{\mathcal{Z}}_N$ such that $z_i^{\text{local nadir}} = (z_{i+1}^1, z_i^2)$. Given a point $\bar{z}_{\text{LP}} = (\bar{z}_{1,\text{LP}}, \bar{z}_{2,\text{LP}})$, which we need to check whether it can be bounded, we compare the objective function value of $\bar{z}_{\text{LP}}$, which is $\bar{z}_{1,\text{LP}} + \bar{z}_{2,\text{LP}}$, with the objective function values of all the local nadir points. Hence, if $\bar{z}_{1,\text{LP}} + \bar{z}_{2,\text{LP}} > \max_{i \in \bar{\mathcal{Z}}_N}(z_{i+1}^1 + z_i^2)$, then we can fathom point $\bar{z}_{\text{LP}}$.

5.3. Integer Fathoming

Given a (mixed) binary solution $(\bar{x}_{\text{LP}}, \bar{y}_{\text{LP}})$, where $\bar{x}_{\text{LP}}$ is binary, the corresponding point in criterion space $\bar{z}_{\text{LP}} = (z_1(\bar{x}_{\text{LP}}, \bar{y}_{\text{LP}}), z_2(\bar{x}_{\text{LP}}, \bar{y}_{\text{LP}}))$ *could* be part of the current set of nondominated points $\bar{\mathcal{Z}}_N$. If this is the case, it is added to the current set of nondominated points $\bar{\mathcal{Z}}_N$. *However, the solution cannot be fathomed* because a later branch may lead to another solution $(\hat{x}_{\text{LP}}, \hat{y}_{\text{LP}})$, where $\hat{x}_{\text{LP}}$ is binary and with the property that $\hat{z}_{\text{LP}} = (z_1(\hat{x}_{\text{LP}}, \hat{y}_{\text{LP}}), z_2(\hat{x}_{\text{LP}}, \hat{y}_{\text{LP}}))$ belongs to the final set of nondominated points, even though the current point $\bar{z}_{\text{LP}}$ does not. Hence, the BOBB algorithm performs what we call *integer branching*. Given the current mixed binary solution $(\bar{x}_{\text{LP}}, \bar{y}_{\text{LP}})$, the binary variables $\bar{x}_{\text{LP}}$ are added to the set of previously seen binary solutions I. Then a new constrained LP problem is formed. First, the previous branching decisions F_0 and F_1 are added, i.e., the binary values are fixed according to F_0 and F_1. For each previously seen binary solution stored in I, the *integer branching constraints* are added; see Equation (10):

$$\sum_{j=1}^{n}\{x_j \mid \bar{x}_j = 0\} + \sum_{j=1}^{n}\{1 - x_j \mid \bar{x}_j = 1\} \geq 1, \quad \forall\, \bar{x} \in I. \quad (10)$$

Formula (10) is sometimes called a no-good constraint; see Hooker (2000). Formulating a no-good constraint for MIP models with nonbinary integer variables is possible, but requires the use of at least one extra variable per no-good constraint. Although this is in general not a problem, it becomes a technical issue when using CPLEX callback functions. CPLEX callback functions do not allow addition of new variables. To circumvent this, the variables have to be preallocated. Preallocation of variables will increase the problem size, and there is no way to guarantee to have enough variables, since it is not possible to predict how many are necessary.

The augmented problem is then solved using the LP solver. Finally, the solution six-tuple for the augmented problem is entered into the enumeration tree. In §§5.8 and 5.9, we discuss a few possible improvements that may make the binary fathoming more efficient.

5.4. Standard Branching

If a solution cannot be infeasibility fathomed, bound fathomed, or binary fathomed, the BOBB algorithm applies standard variable branching.

5.5. The Entire Algorithm

In Figure 7, we present the entire BOBB algorithm in pseudocode. In Figure 7, line 2 establishes the initial nondominated set, and line 3 solves the root problem and generates a six-tuple, which is inserted as the first (partial) solution in the enumeration tree data structure S in line 4. In line 5, the main loop is entered. The

Figure 7 The Biobjective Branch and Bound Algorithm

```
 1 procedure Biobjective Branch and Bound()
 2    Z̄_N = {(0, 1), (1, 0)};
 3    (z̄_LP, x̄_LP, { }, { }, F/IF, { }) ←
          OPT(−, −, { }, { }, −, { });
 4    S = {(z̄_LP, x̄_LP, { }, { }, F/IF, { })};
 5    while (S ≠ { }) do
 6       (z̄_LP, x̄_LP, F_0, F_1, F/IF, I) ← S
 7       if (F/IF = IF) then CONTINUE;
 8       if (z̄_LP ≥ WORST_local nadir) CONTINUE;
 9       if (BINARY(x̄_LP)) then
10          if(NonDom(z̄_LP)) then
                Z̄_N = Z̄_N ∪ {z̄_LP};
11          I = I ∪ {x̄_LP};
12          (z̄_LP, x̄_LP, F_0, F_1, F/IF, I) ←
                OPT(−, −, F_0, F_1, −, I);
13          S ← S ∪ (z̄_LP, x̄_LP, F_0, F_1, F/IF, I);
14          CONTINUE;
15       end if
16       FRACTIONAL(x̄_LP) → x̄_i,LP;
17       (z̄_LP, x̄_LP, F_0, F_1, F/IF, I) ←
             OPT(−, −, F_0 ∪ i, F_1, −, I);
18       S = S ∪ (z̄_LP, x̄_LP, F_0, F_1, F/IF, I);
19       (z̄_LP, x̄_LP, F_0, F_1, F/IF, I) ←
             OPT(−, −, F_0, F_1 ∪ i, −, I);
20       S ← S ∪ (z̄_LP, x̄_LP, F_0, F_1, F/IF, I);
21    end while
22    Z_N = Z̄_N;
23 end procedure
```

main loop is only exited if there are no more intermediate solutions left in the enumeration tree data structure S. In line 6, an intermediate solution is retrieved from S; this solution is represented by a six-tuple. If the intermediate solution is infeasible, it is fathomed in line 7. If it is worse than the worst local nadir point in the current nondominated set, it is fathomed in line 8. If the intermediate solution is feasible ($\bar{x}_{\mathrm{LP}}$ is binary), the corresponding point $\bar{z}_{\mathrm{LP}}$ in criterion space is added to the current set of nondominated points in line 10, provided that it is not dominated by one or more existing points in the current nondominated set. In line 11, the new integer solution is added to the set of integer solutions. In line 12, a new intermediate point is found, using an LP solver, with the added integer constraint. The new solution is added to S in line 13. If the intermediate solution is neither infeasible, bounded, or binary, standard binary branching is performed in lines 16–20. First the index i of the branching variable is found in line 16. Then the two solutions with variable i fixed to 0 and 1, respectively, are added to S.

5.6. Algorithm Efficiency

As noted in §4, problem (7) is very hard to solve. The proposed BOBB algorithm above has some weaknesses compared with the standard single-objective branch and bound algorithm:

- the bounding is significantly weaker, and
- integer solutions cannot be fathomed.

Particularly, the weaker bounding possibilities are problematic. It is well known that the success of an algorithm to solve an MIP problem rests almost exclusively on the tightness of the bound, i.e., the *gap* between the optimal MIP solution and the relaxed (root) LP-optimal solution value.

5.7. Slicing

Unfortunately, bounding in the basic BOBB algorithm is weak, as described in §5.6. Since only bounding against the worst local nadir point is possible, even (fractional) solutions that are dominated by many points in the current set of nondominated points $\bar{\mathcal{Z}}_N$ cannot be fathomed. Applying *slicing* is an attempt to improve the bounding. Suppose that we divide the square with extreme points $(0,0)$, $(1,0)$, $(1,1)$, and $(0,1)$ into s slices. This means that we draw $s-1$ lines originating from the extreme point $(0,0)$. The angle between the z_2-axis and the first line is $90°/s$, the angle between the z_2-axis and the second line is $90 \cdot 2°/s$, and the angle between the z_2-axis and the last line is $(90 \cdot (s-1))°/s$.

In Figures 8(a)–8(d), we assume that we are in the process of searching for nondominated points. The square has been divided into one, two, three, and four slices, respectively. The light grey area has been searched so far, and in addition to z^{UL} and z^{LR}, three additional nondominated points, A, B, and C, have been found. The level curve of the parametric objective function at the current point C is shown with a dotted line.

In Figure 8(a), the square has not been divided into smaller slices (so there is only one slice). We can stop searching for nondominated points when the parametric value of the nondominated points is equal to the parametric value of the worst local nadir point (the local nadir point that has the largest value of the parametric value of all the local nadir points; this point is marked with a small square). At the current point C, we still need to search the white areas for nondominated points. Note that the medium grey part of the area containing dominated points is also searched, whereas the dark grey part is never searched.

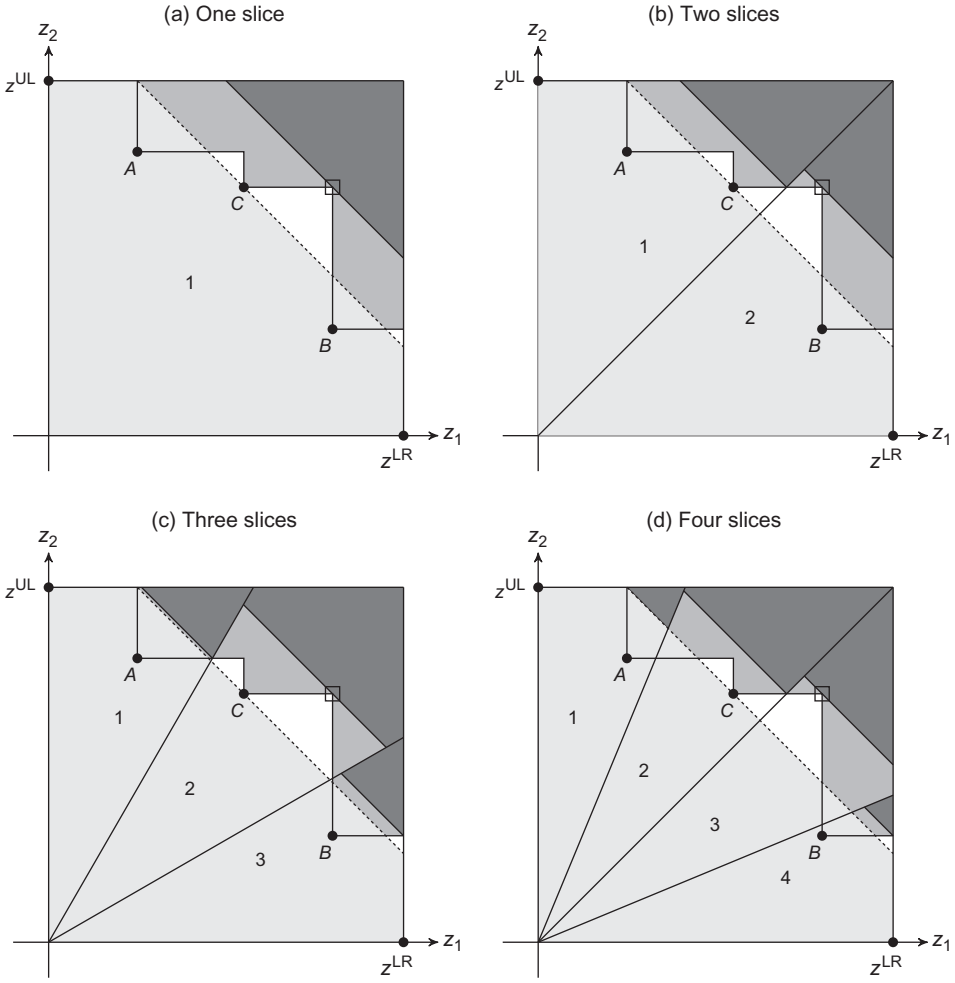
In Figure 8(b), the square has been divided into two slices. In the first slice, marked with 1, a smaller part of the dominated area is searched, compared with Figure 8(a). The search in the first slice stops when the part of the white area in that slice has been searched. The medium grey part in the first slice, which contains only dominated points, is also searched. Note that the upper bound in slice one is better (lower) than the upper bound in that area used when the square has not been divided into smaller slices.

In Figure 8(c), the square has been divided into three slices, and in Figure 8(d), the square has been divided into four slices. The parts of the dominated areas that are searched in each slice are coloured light grey or medium grey, and the parts that are not searched are coloured dark grey. As before, the search in each slice stops whenever the white area belonging to the slice has been searched. The border between the medium grey and the dark grey areas indicates when the search stops. (In each slice the level curve of the parametric objective function is parallel with the borderline between the medium grey and the dark grey areas.) Note that when more slices are introduced, the upper bound on the parametric objective function value within each slice gets smaller (or has the same value). Because of construction, this is true in general.

In each slice, a BOBB algorithm is executed. However, the improved upper bound within each slice comes at a price: It is likely that many of the same intermediate points are handled in different slices, i.e., more work is performed.

Introducing slices into the model is easy. Suppose we divide the square with extreme points $(0,0)$, $(1,0)$, $(1,1)$, and $(0,1)$ into s slices, by drawing $s-1$ lines originating from the extreme point $(0,0)$. We know that the angle between two consecutive lines (including the z_1-axis and the z_2-axis) is $90°/s$. This is denoted the angle size. Let Δ denote

Figure 8 Improving the Upper Bound by Slicing



half the angle size, that is, $\Delta = 0.5 \cdot (90°/s)$. For each slice $i = 1, 2, \ldots, s$, we denote by α_i the center angle as $\alpha_i = (s - i) \cdot (90°/s) + \Delta$.

If we denote the z_2-axis as line 0, the line next to the z_2-axis as line $1, \ldots,$ and the z_1-axis as line s, a specific slice i between two lines $i-1$ and i can be described using the center angle α_i and the angle size $2 \cdot \Delta$. The angle between the two lines is the angle size $2 \cdot \Delta$. The center angle α_i is the average of the two angles between the z_1-axis and the two lines $i-1$ and i. This implies that the angle between the z_1-axis and line i is equal to $\alpha_i - \Delta$, and the angle between the z_1-axis and line $i-1$ is equal to $\alpha_i + \Delta$.

To introduce slice i in the model, we need to add two more constraints:

$$z_2 \geq \tan(\alpha_i - \Delta) \cdot z_1, \tag{11}$$

$$z_2 \leq \tan(\alpha_i + \Delta) \cdot z_1. \tag{12}$$

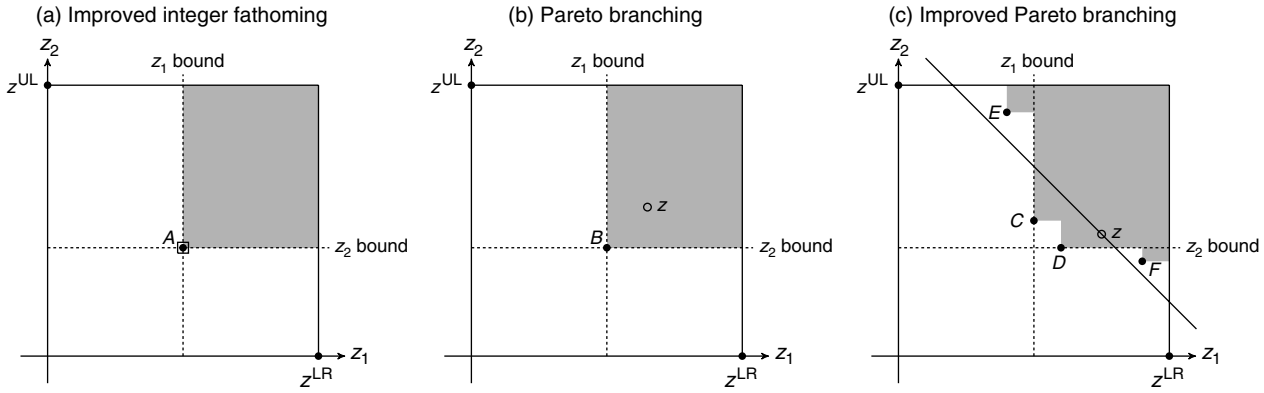
The algorithm shown in Figure 7 can easily be changed to work with slices: Each slice is an almost independent process. When feasible points are found, they are inserted into a joint data structure for all the slices. This means that finding a feasible point in one slice may change the worst local nadir point in another slice.

It seems obvious to parallelize the sliced BOBB algorithm, given that only limited communication between the different slices is necessary. This is, however, left for future research.

5.8. Improved Integer Fathoming

The integer fathoming approach described in §5.3 can be improved by looking at the effect in objective space; see Figure 9(a). In Figure 9(a), point $A = (z_1(x, y), z_2(x, y))$ corresponding to a feasible solution (x, y) to (9) is illustrated as the filled circle. The effect of the branching constraint, given by Equation (10), is to render a small area around the point infeasible. This is illustrated as the box around the circle. The addition of the branching constraints will obviously reduce the remaining feasible space, but since it is constructed exactly such that only

Figure 9 Improved Integer and Pareto Branching



one integer solution is made infeasible, the effects on the objective space are probably minor. On the other hand, we know that a certain part of the objective space becomes dominated, and hence any solution found in this area is not interesting. This fact can be used in the algorithm in the following way: instead of just adding one branching constraint, given by Equation (10), we perform a binary branch where each of the branch children has the branch constraint and a changed upper bound on either z_1 or z_2. In this way, none of the future solutions in the tree below the current integer solution can end up in the dominated area, illustrated by the medium grey area of Figure 9(a).

5.9. Pareto Branching

The improved integer fathoming described in §5.8 should improve the bounding whenever a feasible point is found. In the following discussion we assume that point $z = (z_1(x, y), z_2(x, y))$, where x is fractional.

We first assume that z is dominated by the current Pareto front, but not worse than the current worst local nadir point. In this case, we can perform a special kind of branching, which we term Pareto branching; see Figure 9(b). In Figure 9(b), we assume a nondominated set consisting of three points, the two extreme points z^{UL} and z^{LR}, and one further point B, shown as a filled circle. The current point z is illustrated as an open circle, and this point cannot be bounded, since the parametric objective function value of the worst local nadir point is worse than the parametric objective function value of z. But z is currently in a part of the objective space that is dominated. Hence, we can perform a Pareto branch: simply create two children, one that has a new upper bound on z_1 (the z_1-value of B) and one that has a new upper bound on z_2 (the z_2-value of B). The no-good constraint, given by Equation (10), is, on the other hand, not necessary.

It may be the case that z is dominated by one or several points in the current set of nondominated points. In that case, the strength of the branch constraints can be improved; see Figure 9(c). In Figure 9(c), z is dominated by point D. Point z, however, has a bound on the parametric objective function value lower than the worst local nadir point and cannot be fathomed. Both points C and D dominate that bound and we can perform Pareto branching around both points, but actually the bounds can be strengthened. Instead of choosing a point in the existing Pareto front to branch on, we can move in both directions and check when the current lower bound hits the "wall" of the dominated area. This means that we can use the z_1 bound of C and the z_2 bound of D.

6. Tests

The BOBB algorithm presented in §5 is a *general* algorithm for solving a large subset of biobjective mixed integer programming (BOMIP) problems, as defined in §4. Hence, testing the developed algorithm requires measuring performance on a significant number of MOMIP problems *and* comparing the performance with the generic two-phase algorithm, presented in §3.1.

In this section, we compare the performance of the BOBB algorithm with the two-phase algorithm on a set of six classes of biobjective mixed integer programming problems:

- biobjective assignment problem (BIAP; see Appendix A);
- biobjective knapsack problem (KNAPSACK; see Appendix B);
- biobjective set-covering problem (SET-COVERING; see Appendix C);
- biobjective facility-location problem (FACILITY-LOCATION; see Appendix D);
- biobjective fixed-charge network design problem (FIXED-CHARGE; see Appendix E);
- biobjective random problem (RANDOM; see Appendix F).

For each problem, 300 seconds are allowed for the program to find the set of nondominated points $\mathcal{X}_N$. The setup of each of these problems is described in detail in Appendices A–F. For each problem we generate a number of different sized problems: BIAP 10, KNAPSACK 11, SET-COVERING 8, FACILITY-LOCATION 6, FIXED-CHARGE 9, RANDOM 16. For each problem size we generate 15 samples. This leads to a total of 900 BOMIP models.

We test six different versions of the BOBB algorithm, each with a different number of slices: 1, 2, 4, 8, 16, and 32. Each algorithm, hereafter named BOBB-1, etc., is tested in two versions, one *without* Pareto branching and one *with* Pareto branching; see §5.9. Furthermore, we test the generic two-phase algorithm on the same samples. In total, 13 algorithms are tested on the 900 BOMIP models, leading to a total of 11,700 tests.

The algorithms are implemented in C++ using the g++ (GCC) compiler, version 4.1.2 (Red Hat 4.1.2-48). The two-phase algorithm is implemented using the ILOG CPLEX 12.1.1 callable library. The BOBB algorithm is implemented using the ILOG CPLEX 12.1.1 callback routines.

The tests are performed on a cluster of Linux machines using up to 64 machines in parallel, each machine consisting of two Xeon X5550 2.67 GHz quadcore CPUs with 8 MB of cache and 24 GB, 1,333 MHz DDR3 ECC ram. All of the programs are executed in single threads, meaning that none of our algorithms are parallelized, but the cluster makes it possible to perform the 11,700 tests in a reasonable time. Using a batch system, it is ensured that all tests are performed with exclusive processor rights, meaning that the programs cannot be interrupted or preempted by other processes.

In the remainder of this section, we test the importance of Pareto branching in §6.1. Pareto branching turns out to be absolutely essential for the BOBB algorithms; hence, we only consider BOBB algorithms with Pareto branching. In §6.2, the average solution times and average solution success ratio for all algorithms (with Pareto branching) and for all six problem classes are presented. We calculate the success ratio of an algorithm for a specific test as the number of times the Pareto-optimal front was found in less than 300 seconds divided by the number of tests (15). Some overall comments on the results in §6.2 are given in §6.2.7. Finally, in §6.2.8, we compare the results from the two-phase algorithm with the results from the BOBB-32 algorithm to explain why our implementation of the generic two-phase method cannot compete on five of the six problem classes.

6.1. Tests With Pareto Branching vs. Tests Without Pareto Branching

During testing, Pareto branching turned out to be essential. None of the BOBB algorithms were competitive without Pareto branching. An example of this can be seen in Figures 10(a) and 10(b), where the six different BOBB algorithms are compared with the BIAP in two versions, with and without Pareto branching. It can be seen that no algorithm without Pareto branching can solve BIAPs with more than 200 binary variables, whereas all algorithms with Pareto branching can solve BIAPs with more than 600 binary variables (within the time limit 300 seconds). The situation is the same for all the other problem classes. For this reason, in the following we consider only BOBB algorithms with Pareto branching.

6.2. Tests With Pareto Branching

For all the remaining tests, we use Pareto branching for the BOBB algorithms. The six problem classes are described in Appendices A–F. The first column (Sz.) in each of the Figures 11(a)–16(a) is the number of binary variables. For each of the seven algorithms tested, we report the CPU time in seconds (Time) and the success ratio (Suc.). All numbers in the tables in this section are averaged over 15 samples for each problem size.

6.2.1. BIAP. The BIAP is tested using 150 biobjective assignment problems, with a size between

Figure 10 Tests With Pareto Branching vs. Tests Without Pareto Branching

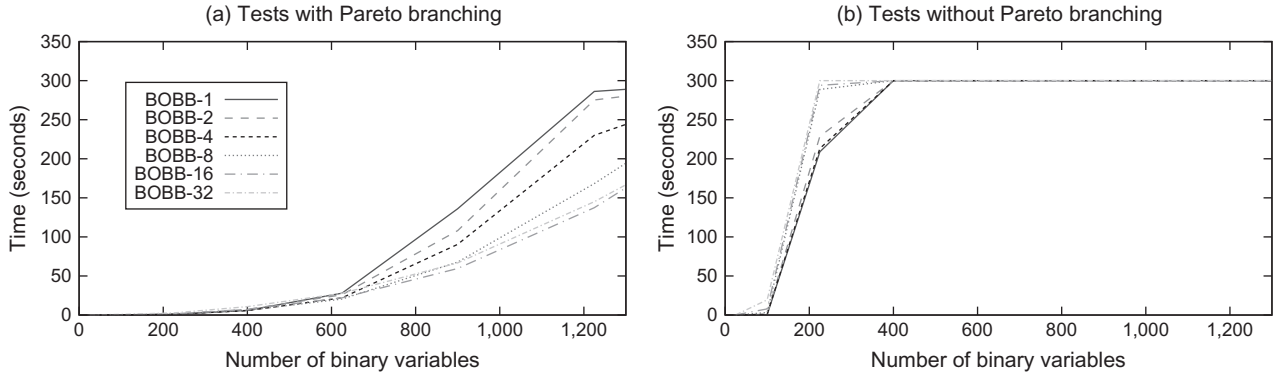
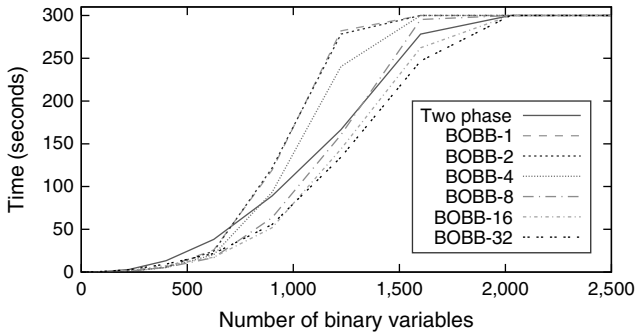


Figure 11 Performance of Seven Algorithms on BIAP

(a) Performance of seven algorithms on BIAP

	BOBB-1		BOBB-2		BOBB-4		BOBB-8		BOBB-16		BOBB-32		Two-phase	
Sz.	Time	Suc.	Time	Suc.	Time	Suc.	Time	Suc.	Time	Suc.	Time	Suc.	Time	Suc.
25	0.00	1.00	0.00	1.00	0.00	1.00	0.00	1.00	0.10	1.00	0.10	1.00	0.00	1.00
100	0.10	1.00	0.10	1.00	0.10	1.00	0.20	1.00	0.40	1.00	0.70	1.00	0.50	1.00
225	0.90	1.00	0.90	1.00	1.00	1.00	1.20	1.00	1.90	1.00	2.70	1.00	3.00	1.00
400	5.70	1.00	5.90	1.00	5.30	1.00	5.40	1.00	7.50	1.00	9.40	1.00	13.30	1.00
625	25.90	1.00	24.50	1.00	20.40	1.00	17.00	1.00	17.40	1.00	21.70	1.00	38.30	1.00
900	119.20	1.00	121.70	1.00	93.20	1.00	63.70	1.00	51.90	1.00	55.60	1.00	89.10	1.00
1,225	282.10	0.27	278.30	0.33	240.30	0.80	160.50	0.93	143.90	0.93	134.30	1.00	166.60	1.00
2,025	300.00	0.00	300.00	0.00	300.00	0.00	295.40	0.13	262.30	0.47	246.80	0.80	278.20	0.53
2,500	300.00	0.00	300.00	0.00	300.00	0.00	300.00	0.00	300.00	0.00	300.00	0.00	300.00	0.00

(b) Time



(c) Success ratio

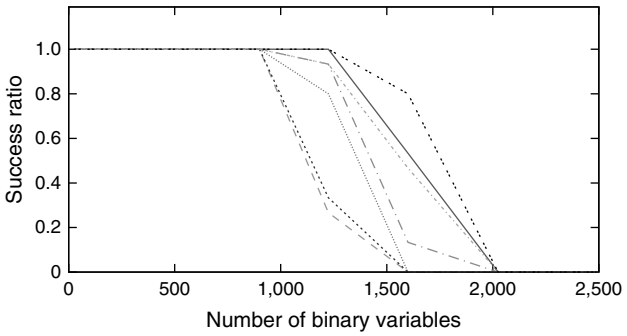
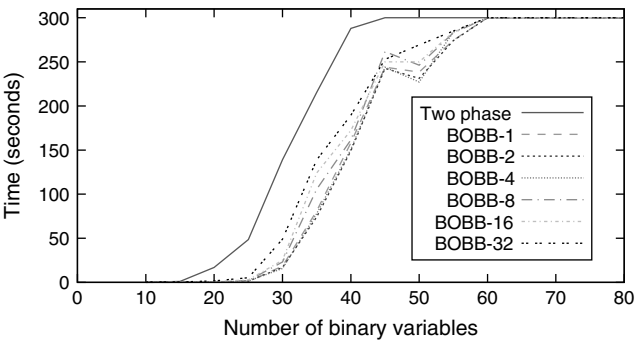


Figure 12 Performance of Seven Algorithms on KNAPSACK

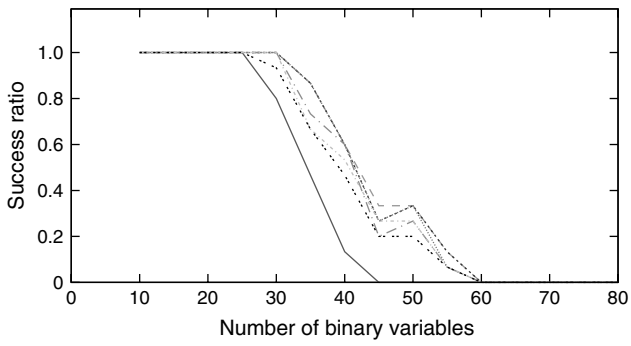
(a) Performance of seven algorithms on KNAPSACK

	BOBB-1		BOBB-2		BOBB-4		BOBB-8		BOBB-16		BOBB-32		Two-phase	
Sz.	Time	Suc.	Time	Suc.	Time	Suc.	Time	Suc.	Time	Suc.	Time	Suc.	Time	Suc.
10	0.00	1.00	0.00	1.00	0.00	1.00	0.10	1.00	0.10	1.00	0.40	1.00	0.10	1.00
15	0.00	1.00	0.10	1.00	0.10	1.00	0.10	1.00	0.20	1.00	0.40	1.00	1.00	1.00
20	0.40	1.00	0.40	1.00	0.40	1.00	0.60	1.00	0.80	1.00	1.70	1.00	16.90	1.00
25	1.10	1.00	1.30	1.00	1.50	1.00	1.90	1.00	2.90	1.00	5.30	1.00	48.50	1.00
30	18.40	1.00	17.00	1.00	15.70	1.00	23.10	1.00	25.10	1.00	48.90	0.93	139.00	0.80
35	80.80	0.87	74.10	0.87	77.80	0.87	105.50	0.73	123.90	0.67	138.70	0.67	215.20	0.47
40	158.30	0.60	149.20	0.60	151.00	0.60	161.50	0.60	172.30	0.53	189.60	0.47	287.90	0.13
45	244.00	0.33	242.60	0.27	244.00	0.27	261.70	0.20	250.00	0.27	252.90	0.20	300.00	0.00
50	238.40	0.33	231.20	0.33	227.10	0.33	246.20	0.27	250.00	0.27	269.20	0.20	300.00	0.00
55	274.50	0.13	274.00	0.13	283.70	0.07	282.90	0.07	283.60	0.07	284.90	0.07	300.00	0.00
60	300.00	0.00	300.00	0.00	300.00	0.00	300.00	0.00	300.00	0.00	300.00	0.00	300.00	0.00

(b) Time



(c) Success ratio



25 binary variables and 2,500 binary variables. The results for the seven algorithms are given in Figure 11(a) and they are visualized in Figures 11(b) and 11(c). It can be seen that the results obtained using BOBB-16 and BOBB-32 are better than the results obtained by the generic two-phase algorithm, and that BOBB-32 is the best.

6.2.2. KNAPSACK. KNAPSACK is tested using 165 biobjective knapsack problems, with a size between 10 binary variables and 60 binary variables. The results for the seven algorithms are given in Figure 12(a), and they are visualized in Figures 12(b) and 12(c). It can be seen that all the BOBB algorithms perform better than the generic two-phase algorithm and that BOBB-2 is the best algorithm.

6.2.3. SET-COVERING. SET-COVERING is tested using 120 biobjective set-covering problems, with a size between 50 binary variables and 400 binary variables. The results for the seven algorithms are given in Figure 13(a), and they are visualized in Figures 13(b) and 13(c). It can be seen that the generic two-phase algorithm performs best. The runner up is the BOBB-32 algorithm.

6.2.4. FACILITY-LOCATION. FACILITY-LOCATION is tested using 90 biobjective facility-location problems, with a size between 5 binary variables and 35 binary variables. The results for the seven algorithms are given in Figure 14(a), and they are visualized in Figures 14(b) and 14(c). It can be seen that all the BOBB algorithms perform better than the generic two-phase algorithm, and that BOBB-2 is the best algorithm.

6.2.5. FIXED-CHARGE. FIXED-CHARGE is tested using 135 biobjective fixed-charge problems, with a size between 3 binary variables and 55 binary variables. The results for the seven algorithms are given in Figure 15(a), and they are visualized in Figures 15(b) and 15(c). It can be seen that all the BOBB algorithms perform better than the generic two-phase algorithm and that BOBB-8 is the best algorithm.

6.2.6. RANDOM. RANDOM is tested using 240 biobjective random problems, with a size between 5 binary variables and 80 binary variables. The results for the seven algorithms are given in Figure 16(a), and they are visualized in (b) and (c). It can be seen that BOBB-32 is the best algorithm.

6.2.7. Overall Comments to Results. From the results in §§6.2.1–6.2.6, we can see that the generic two-phase algorithm is the fastest algorithm only for SET-COVERING. It can be debated which BOBB algorithm is the better, but a strong contender is BOBB-32.

Figure 13 Performance of Seven Algorithms on SET-COVERING

(a) Performance of seven algorithms on SET-COVERING

	BOBB-1		BOBB-2		BOBB-4		BOBB-8		BOBB-16		BOBB-32		Two-phase	
Sz.	Time	Suc.	Time	Suc.	Time	Suc.	Time	Suc.	Time	Suc.	Time	Suc.	Time	Suc.
50	0.00	1.00	0.10	1.00	0.10	1.00	0.10	1.00	0.20	1.00	0.30	1.00	0.10	1.00
100	1.80	1.00	1.90	1.00	1.70	1.00	2.10	1.00	2.50	1.00	3.30	1.00	0.50	1.00
150	24.10	1.00	25.70	1.00	20.10	1.00	17.20	1.00	15.50	1.00	20.10	1.00	3.50	1.00
200	149.00	0.80	158.30	0.73	145.40	0.80	110.50	0.93	94.70	0.93	97.60	0.93	28.30	1.00
250	273.90	0.27	285.50	0.20	271.40	0.20	247.20	0.33	227.90	0.40	218.80	0.47	120.00	0.87
300	295.30	0.07	300.00	0.00	295.40	0.07	293.10	0.07	293.60	0.07	286.50	0.13	244.70	0.40
350	300.00	0.00	300.00	0.00	300.00	0.00	300.00	0.00	300.00	0.00	300.00	0.00	284.90	0.13
400	300.00	0.00	300.00	0.00	300.00	0.00	300.00	0.00	300.00	0.00	300.00	0.00	300.00	0.00

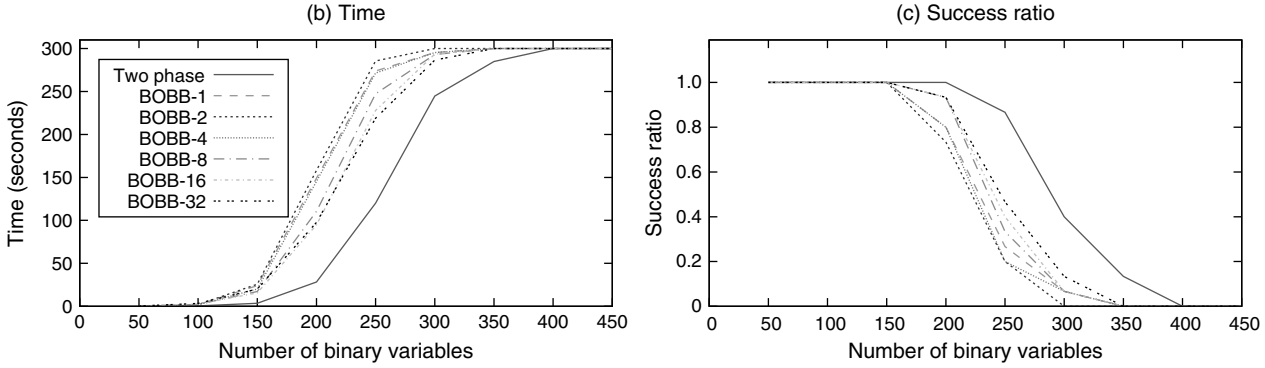
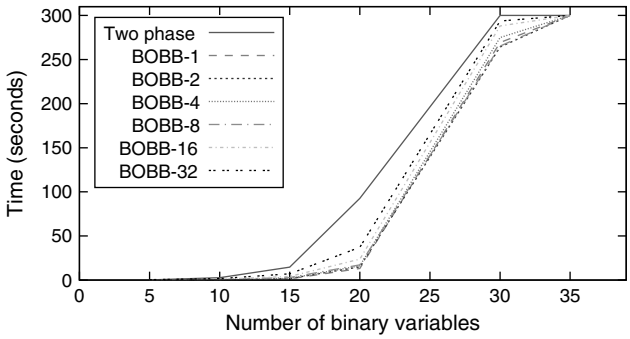


Figure 14 Performance of Seven Algorithms on FACILITY-LOCATION

(a) Performance of seven algorithms on FACILITY-LOCATION

	BOBB-1		BOBB-2		BOBB-4		BOBB-8		BOBB-16		BOBB-32		Two-phase	
Sz.	Time	Suc.	Time	Suc.	Time	Suc.	Time	Suc.	Time	Suc.	Time	Suc.	Time	Suc.
5	0.00	1.00	0.00	1.00	0.10	1.00	0.10	1.00	0.20	1.00	0.40	1.00	0.10	1.00
10	0.10	1.00	0.20	1.00	0.30	1.00	0.60	1.00	0.90	1.00	1.70	1.00	2.70	1.00
15	1.40	1.00	1.60	1.00	2.00	1.00	2.80	1.00	4.10	1.00	7.10	1.00	14.60	1.00
20	13.10	1.00	14.90	1.00	16.10	1.00	17.20	1.00	23.70	1.00	37.20	1.00	92.60	1.00
30	269.40	0.33	264.20	0.40	274.90	0.33	265.20	0.40	288.10	0.13	293.70	0.13	300.00	0.00
35	300.00	0.00	300.00	0.00	300.00	0.00	300.00	0.00	300.00	0.00	300.00	0.00	300.00	0.00

(b) Time



(c) Success ratio

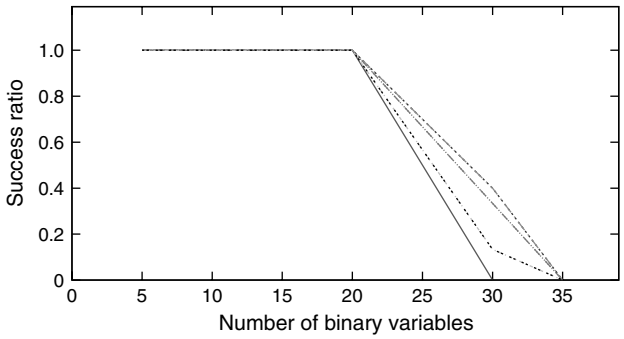
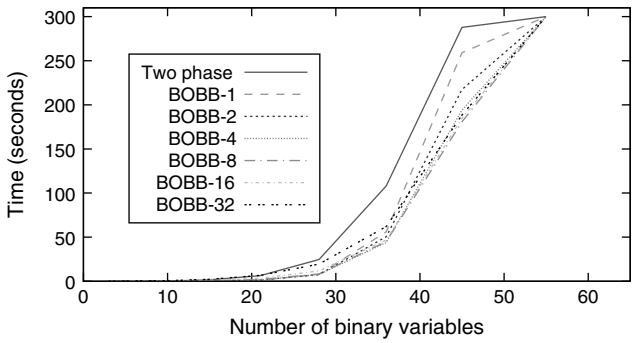


Figure 15 Performance of Seven Algorithms on FIXED-CHARGE

(a) Performance of seven algorithms on FIXED-CHARGE

	BOBB-1		BOBB-2		BOBB-4		BOBB-8		BOBB-16		BOBB-32		Two-phase	
Sz.	Time	Suc.	Time	Suc.	Time	Suc.	Time	Suc.	Time	Suc.	Time	Suc.	Time	Suc.
3	0.00	1.00	0.00	1.00	0.00	1.00	0.00	1.00	0.00	1.00	0.10	1.00	0.00	1.00
6	0.00	1.00	0.00	1.00	0.00	1.00	0.10	1.00	0.10	1.00	0.30	1.00	0.00	1.00
10	0.00	1.00	0.10	1.00	0.10	1.00	0.20	1.00	0.40	1.00	0.80	1.00	0.20	1.00
15	0.20	1.00	0.30	1.00	0.40	1.00	0.60	1.00	1.20	1.00	2.30	1.00	1.60	1.00
21	1.20	1.00	1.30	1.00	1.50	1.00	2.00	1.00	3.40	1.00	6.60	1.00	6.40	1.00
28	7.80	1.00	7.60	1.00	7.60	1.00	8.30	1.00	11.80	1.00	19.50	1.00	24.70	1.00
36	56.00	1.00	50.50	1.00	44.00	1.00	45.10	1.00	46.70	1.00	61.90	1.00	108.10	1.00
45	259.10	0.67	217.40	0.87	194.00	0.87	180.80	0.87	184.80	0.87	188.50	0.87	287.80	0.40
55	300.00	0.00	300.00	0.00	300.00	0.00	300.00	0.00	300.00	0.00	300.00	0.00	300.00	0.00

(b) Time



(c) Success ratio

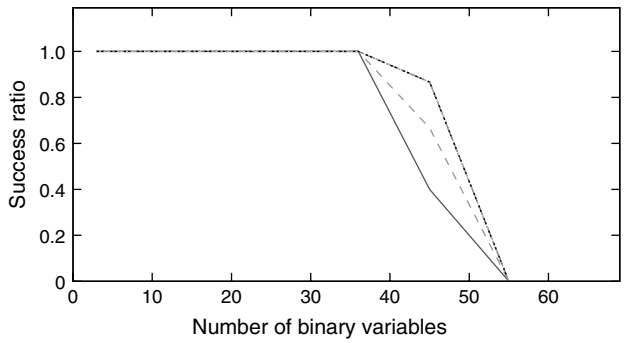
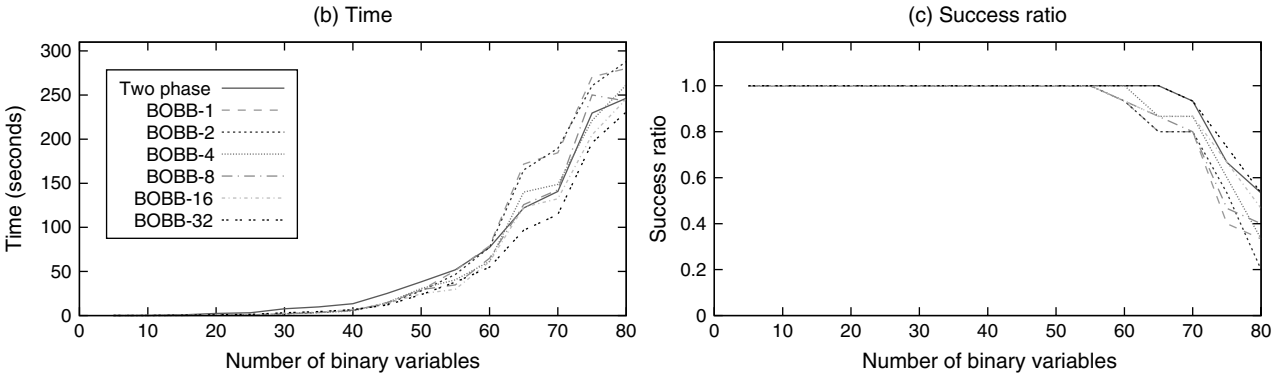


Figure 16 Performance of Seven Algorithms on RANDOM

(a) Performance of seven algorithms on RANDOM

	BOBB-1		BOBB-2		BOBB-4		BOBB-8		BOBB-16		BOBB-32		Two-phase	
Sz.	Time	Suc.	Time	Suc.	Time	Suc.	Time	Suc.	Time	Suc.	Time	Suc.	Time	Suc.
5	0.00	1.00	0.00	1.00	0.00	1.00	0.00	1.00	0.00	1.00	0.10	1.00	0.00	1.00
10	0.00	1.00	0.00	1.00	0.00	1.00	0.10	1.00	0.10	1.00	0.20	1.00	0.10	1.00
15	0.10	1.00	0.10	1.00	0.10	1.00	0.20	1.00	0.20	1.00	0.40	1.00	0.70	1.00
20	0.30	1.00	0.40	1.00	0.40	1.00	0.50	1.00	0.60	1.00	0.90	1.00	2.40	1.00
25	0.60	1.00	0.60	1.00	0.70	1.00	0.80	1.00	0.90	1.00	1.40	1.00	3.10	1.00
30	2.10	1.00	2.00	1.00	2.00	1.00	2.10	1.00	2.70	1.00	3.20	1.00	7.70	1.00
35	3.70	1.00	3.60	1.00	3.30	1.00	3.50	1.00	4.10	1.00	4.60	1.00	9.80	1.00
40	6.10	1.00	6.00	1.00	5.10	1.00	5.60	1.00	7.00	1.00	6.70	1.00	13.50	1.00
45	14.00	1.00	13.10	1.00	14.60	1.00	14.60	1.00	13.70	1.00	12.00	1.00	25.10	1.00
50	28.70	1.00	27.90	1.00	30.70	1.00	29.50	1.00	24.70	1.00	23.80	1.00	38.30	1.00
55	51.00	1.00	46.50	1.00	40.50	1.00	34.80	1.00	29.80	1.00	38.00	1.00	51.90	1.00
60	79.10	0.93	77.40	0.93	61.30	1.00	65.00	0.93	61.10	0.93	54.90	1.00	77.20	1.00
65	171.70	0.80	165.10	0.80	139.90	0.87	125.80	0.87	122.90	0.87	96.80	1.00	122.00	1.00
70	184.50	0.80	189.60	0.80	148.80	0.87	142.50	0.80	132.40	0.87	114.50	0.93	140.70	0.93
75	270.50	0.40	260.50	0.53	221.10	0.60	250.10	0.47	205.00	0.67	195.90	0.73	229.60	0.67
80	280.10	0.33	287.90	0.20	261.50	0.33	243.00	0.40	245.50	0.47	230.80	0.53	246.30	0.53



The BOBB-32 algorithm is as good or better than the generic two-phase algorithm, both with respect to average solution time and to the success ratio on five of the six problems, but BOBB-32 is significantly worse than the generic two-phase algorithm on SET-COVERING.

6.2.8. Is the Two-Phase Algorithm Badly Implemented? There can be a number of reasons for the generic two-phase algorithm to be slower than the different BOBB algorithms. A possible explanation could be that the implemented version of the generic two-phase algorithm is programmed such that it is inefficient. To test that, we measured the percentage of the average time spent in the CPLEX solver during the execution of the generic two-phase algorithm; see Table 1. The first column shows the problem name, the second column shows the size of the problem measured in binary variables, and the third column shows the average percentage of the total generic two-phase algorithm running time spend by CPLEX. We have chosen to look at the largest size, for which the generic two-phase algorithm solves all problems to optimality within the time limit. This is to avoid fluctuations for very short execution times and to get a reliable average by having 15 samples to average over. As can be seen, even a dramatic speed up of the non-CPLEX part of the generic two-phase algorithm will only have a very minor effect on the execution time.

Table 1 Average Time Spent by CPLEX During the Two-Phase Algorithm

Problem class	Number of binary variables	Percentage of time spent by CPLEX (%)
BIAP	30	99.9
FACILITY-LOCATION	15	99.9
FIXED-CHARGE	26	97.2
KNAPSACK	20	99.7
RANDOM	55	98.8
SET-COVERING	50	99.9

6.2.9. Why Is the Two-Phase Algorithm Slower? In this section we give a detailed comparison of the BOBB-32 algorithm and the generic two-phase algorithm. The data are given for each test problem in Tables 2–7. In each table, the first column shows the problem size (Sz.), measured in the number of binary variables. The next two columns show the average number of branch nodes (i.e., number of nodes processed by the CPLEX branch and bound algorithm) and average simplex iterations (iter.) for each problem size for the BOBB-32 algorithm. The next eight columns show the average number of Phase 1 iterations, the average Phase 1 execution time, the average number of Phase 2 iterations, the average Phase 2 execution time, the average number of branch nodes, the average number of simplex iterations, the average CPLEX solution time percentage, and the average CPLEX root solution time (root t.), respectively, for the generic two-phase algorithm. Note that all numbers are averages over the the samples that were completed within the time requirements.

The obvious parameter to look at when comparing a branch and bound algorithm with an (iterated) branch and bound algorithm, i.e., the generic two-phase algorithm, is the number of branch nodes processed. On the six test problems, the generic two-phase algorithm requires fewest branch nodes on four problems and BOBB-32 on the remaining two problems. This does not explain the difference in execution times. A possible explanation can be found in Table 6, where the generic two-phase algorithm has (by far) the lowest number of branch nodes, but on the other hand a significantly higher number of simplex iterations. This is probably due to the fact that the generic two-phase algorithm starts from scratch for each iteration and hence has to solve many root nodes, whereas the BOBB-32 only has to solve one. The solution of the root node in a branch and bound algorithm takes significantly more iterations than the other nodes in the branch and bound tree.

Another explanation of the generic two-phase algorithm performance can be found in the last column in the tables: the CPLEX root solution time, i.e., the time it takes before the first branch. Looking at the tables, it becomes clear that the initialization time is critical for BIAP (Table 2) and for FIXED-CHARGE (Table 6). The initialization time has some effect on the solution times for KNAPSACK (Table 3), FACILITY-LOCATION (Table 5), and RANDOM (Table 7). Finally, for SET-COVERING (Table 4), the root solution time is insignificant, and this is the problem where the generic two-phase algorithm performs best. We speculate that the reason for the very

Table 2 Comparison Between BOBB-32 and the Two-Phase Algorithm on BIAP

	BOBB-32		Two-phase							
Sz.	Branch nodes	Simplex iter.	Phase 1 iter.	Phase 1 time	Phase 2 iter.	Phase 2 time	Branch nodes	Simplex iter.	CPLEX %	CPLEX root t.
25	141.30	790.40	4.10	0.00	4.70	0.00	0.00	49.70	76.70	0.00
100	2,384.30	13,056.30	10.50	0.10	20.40	0.50	138.20	2,917.00	97.50	0.50
225	7,790.10	72,012.30	22.50	0.30	49.70	2.70	974.30	16,885.70	99.50	2.70
400	20,447.30	285,925.59	32.10	0.90	95.80	12.40	5,127.50	74,972.60	99.70	10.80
625	35,531.40	523,692.69	42.70	2.10	147.70	36.10	11,774.50	163,428.20	99.80	30.10
900	70,291.30	1,251,906.50	51.30	4.00	205.30	85.10	25,307.50	356,092.91	99.90	66.30
1,225	132,768.30	2,692,812.75	61.90	5.20	285.30	161.30	47,241.70	679,813.88	99.90	116.70
1,600	182,340.80	4,119,906.50	71.80	7.80	310.20	251.20	57,867.60	938,546.00	99.80	181.00

Table 3 Comparison Between BOBB-32 and the Two-Phase Algorithm on KNAPSACK

	BOBB-32		Two-phase							
Sz.	Branch nodes	Simplex iter.	Phase 1 iter.	Phase 1 time	Phase 2 iter.	Phase 2 time	Branch nodes	Simplex iter.	CPLEX %	CPLEX root t.
10	532.70	1,658.30	2.30	0.00	14.50	0.10	66.20	656.20	79.50	0.10
15	2,980.80	6,785.80	4.30	0.00	43.50	0.90	2,536.90	8,682.90	92.60	0.80
20	17,127.90	33,049.90	5.70	0.00	245.90	16.80	232,229.30	501,101.69	98.00	6.40
25	45,672.90	82,626.70	6.50	0.00	569.60	48.40	778,215.12	1,991,254.12	99.70	14.30
30	115,555.40	220,781.80	7.50	0.10	584.30	98.60	1,462,146.50	3,428,489.25	99.80	28.00
35	201,601.09	369,479.50	8.40	0.10	746.60	118.20	1,750,195.62	4,412,314.50	99.80	34.10
40	262,329.69	489,852.59	11.00	0.10	1,172.00	209.20	3,077,031.50	8,122,591.50	99.70	53.60
45	282,993.69	570,177.31	0.00	0.00	0.00	0.00	0.00	0.00	99.70	0.00
50	481,339.00	1,014,827.00	0.00	0.00	0.00	0.00	0.00	0.00	99.70	0.00
55	338,276.00	841,243.00	0.00	0.00	0.00	0.00	0.00	0.00	99.70	0.00

Table 4 Comparison Between BOBB-32 and the Two-Phase Algorithm on SET-COVERING

	BOBB-32		Two-phase							
Sz.	Branch nodes	Simplex iter.	Phase 1 iter.	Phase 1 time	Phase 2 iter.	Phase 2 time	Branch nodes	Simplex iter.	CPLEX %	CPLEX root t.
50	1,908.90	5,665.50	8.10	0.00	7.50	0.00	51.90	428.00	99.30	0.10
100	26,960.30	108,390.90	16.30	0.20	16.20	0.30	2,362.70	11,058.60	99.20	0.30
150	138,594.30	676,477.00	25.80	1.00	25.90	2.40	22,913.90	132,512.09	99.80	0.90
200	499,423.59	2,908,694.00	34.30	8.00	37.00	20.30	163,746.70	1,333,363.88	99.90	1.70
250	696,363.31	5,623,434.50	42.70	22.00	46.10	70.30	405,664.91	4,037,387.75	100.00	2.70
300	917,377.00	8,108,530.00	47.70	39.40	56.30	122.20	568,217.19	6,458,024.00	99.90	4.50
350	0.00	0.00	60.00	63.80	67.00	122.70	615,973.00	9,015,731.00	100.00	5.50

Table 5 Comparison Between BOBB-32 and the Two-Phase Algorithm on FACILITY-LOCATION

	BOBB-32		Two-phase							
Sz.	Branch nodes	Simplex iter.	Phase 1 iter.	Phase 1 time	Phase 2 iter.	Phase 2 time	Branch nodes	Simplex iter.	CPLEX %	CPLEX root t.
5	318.70	5,755.70	7.30	0.00	7.10	0.00	4.10	1,327.60	94.40	0.10
10	3,048.80	37,140.10	19.70	1.00	26.30	1.70	1,015.10	22,878.90	99.40	2.40
15	13,589.30	123,860.10	29.50	4.30	50.00	10.30	9,335.90	121,866.70	99.60	10.70
20	61,045.60	526,599.62	39.70	18.90	82.50	73.50	71,723.70	935,185.69	99.90	44.70
30	226,023.50	2,127,489.50	0.00	0.00	0.00	0.00	0.00	0.00	100.00	0.00

Table 6 Comparison Between BOBB-32 and the Two-Phase Algorithm on FIXED-CHARGE

	BOBB-32		Two-phase							
Sz.	Branch nodes	Simplex iter.	Phase 1 iter.	Phase1 time	Phase 2 iter.	Phase 2 time	Branch nodes	Simplex iter.	CPLEX %	CPLEX root t.
3	9.20	389.10	1.00	0.00	1.00	0.00	0.00	8.40	90.00	0.00
4	274.70	1,772.80	5.50	0.00	4.20	0.00	2.70	656.10	71.10	0.00
5	623.20	6,230.40	11.70	0.10	10.80	0.10	10.20	3,636.10	87.40	0.20
6	1,609.30	21,836.00	20.90	0.40	27.00	1.10	100.10	17,620.60	93.50	1.40
7	3,147.70	60,979.70	31.50	1.40	43.30	4.80	307.30	51,346.20	96.10	5.70
8	5,890.30	149,752.50	43.10	4.60	71.30	19.20	620.50	144,737.59	95.70	21.40
9	14,283.50	358,665.09	56.20	13.20	122.50	92.40	3,745.70	430,482.09	97.20	83.50
10	23,533.80	762,272.00	75.00	39.80	161.30	211.50	2,654.30	814,376.31	94.50	213.50

Table 7 Comparison Between BOBB-32 and the Two-Phase Algorithm on RANDOM

	BOBB-32		Two-phase							
Sz.	Branch nodes	Simplex iter.	Phase 1 iter.	Phase 1 time	Phase 2 iter.	Phase 2 time	Branch nodes	Simplex iter.	CPLEX %	CPLEX root t.
5	22.80	275.90	3.90	0.00	4.30	0.00	0.00	8.70	100.00	0.00
10	690.60	1,255.20	9.40	0.00	13.90	0.10	23.00	375.70	74.40	0.10
15	2,775.60	5,197.50	16.50	0.10	33.10	0.50	365.50	2,721.30	90.40	0.60
20	7,684.70	14,513.40	22.50	0.30	58.80	2.00	2,344.90	12,828.70	96.40	1.90
25	11,589.00	22,305.10	26.90	0.30	80.60	2.70	2,236.80	14,637.60	96.10	2.50
30	27,485.60	53,219.30	34.20	0.60	139.60	7.00	9,280.00	52,190.30	96.90	5.70
35	35,473.90	74,284.20	38.10	0.70	147.10	8.90	14,101.70	81,774.50	97.40	6.60
40	47,818.40	98,210.50	43.30	0.90	179.10	12.30	16,153.10	99,903.90	97.30	9.20
45	74,221.70	163,229.50	50.70	1.50	238.80	23.10	31,294.90	189,288.59	97.70	16.20
50	129,250.10	302,720.91	56.90	2.30	271.80	35.10	66,536.10	382,748.09	97.20	19.90
55	155,782.41	356,204.19	60.70	2.70	363.30	48.40	77,887.90	446,607.69	98.10	28.70
60	242,267.00	583,937.88	65.10	3.60	385.70	70.70	139,106.00	778,472.88	97.10	35.50
65	383,210.09	898,891.50	72.20	3.80	529.80	116.90	186,822.30	1,026,329.62	98.80	59.80
70	370,297.91	866,256.88	77.00	6.50	551.50	118.30	199,823.80	1,136,221.88	97.20	61.80
75	480,337.91	1,200,931.50	83.40	8.50	598.90	179.50	230,486.70	1,382,663.75	96.60	87.60
80	515,813.81	1,217,430.25	87.80	8.50	651.20	183.50	249,936.91	1,472,929.12	94.90	87.70

fast root solution time is that the problems are solved by the preprocessor. Since we are using the CPLEX propriatory code, we can, however, not confirm this.

7. Conclusions

In this paper, we have presented a survey of the most recent methods for generating the full set of nondominated points for MOMIP models. Furthermore, we have devised a new approach to extend the classic branch and bound algorithm in such a way that it can handle an important subclass of MOMIP problems. The developed BOBB algorithm is tested in 12 variants and compared with a generic two-phase algorithm. It is found that the proposed branch and bound algorithm, in particular the BOBB-32 algorithm, in general performs better than the generic two-phase algorithm (with our implementation), except for one class of biobjective problems. To the best of our knowledge, this is the first time that an algorithm, which can find the full set of nondominated points for BOMIP problems, has been devised which is competitive with a two-phase algorithm. We furthermore explain the performance gap with a combination of the generic two-phase algorithm's number of branch nodes and simplex iterations and with its number of root solves. We have further argued that this is a very important result, since it opens up for inclusion of a host of OR techniques that today cannot be applied to the two-phase algorithm. However, the inclusion of these other OR techniques is left for future research.

Acknowledgments

The authors thank department editor Dimitris Bertsimas, the associate editor, and two anonymous referees for their constructive comments and suggestions that helped improve this paper considerably.

Appendix A. Biobjective Assignment Problem

The biobjective assignment problem is the simplest possible multiobjective extension of the classical assignment problem. The problem is to assign n persons to n jobs so that some objectives are optimized. The problem description below is taken from Przybylski et al. (2008).

A.1. Indexes

n: Number of rows. This is also the number of columns.

A.2. Data

$c^1_{ij} \in [0, 20]$: Costs of first objective function.

$c^2_{ij} \in [0, 20]$: Costs of second objective function.

The two costs are generated uniformly in the respective intervals.

A.3. Variables

$$x_{ij} = \begin{cases} 1 & \text{if person } i \text{ is assigned to job } j, \\ 0 & \text{otherwise.} \end{cases}$$

A.4. Mathematical Model

$$\begin{aligned} \min \ & \left(\sum_{i=1}^{n} \sum_{j=1}^{n} c^1_{ij} x_{ij}, \sum_{i=1}^{n} \sum_{j=1}^{n} c^2_{ij} x_{ij} \right) \\ \text{s.t.} \ & \sum_{i=1}^{n} x_{ij} = 1, \quad j = 1, \ldots, n; \\ & \sum_{j=1}^{n} x_{ij} = 1, \quad i = 1, \ldots, n; \\ & x_{ij} \in \{0, 1\}, \quad i, j = 1, \ldots, n. \end{aligned} \tag{A1}$$

A.5. Test problems

We generate 10 different sized problems with $n = 5, 10, 15, \ldots, 50$. The number of binary variables is in the range $50 \le n^2 \le 2{,}500$.

Appendix B. Biobjective Knapsack Problem

This is the classical knapsack problem, but with two objectives. We are given a knapsack with capacity b and a set of n items with weights a_j, $j = 1, \ldots, n$. The problem is to determine which items to put into the knapsack, so as to minimize the two objectives.

B.1. Indexes

n: The number of items (binary variables).

B.2. Data

$c^1_j \in [-9, -1]$: Negative cost of first objective of item j.

$c^2_j = -1 \cdot (10 + c^1_j)$: Negative cost of second objective of item j.

$a_j \in [1, 10]$: Weight of item j.

$b = \frac{1}{2} \cdot \sum_{j=1}^{n} a_j$: The total acceptable weight available in the knapsack.

The numbers c^1_j are generated uniformly in the interval $[-9, -1]$.

Note that the two costs c^1_j and c^2_j are negatively correlated.

B.3. Variables

$$x_j = \begin{cases} 1 & \text{if item } j \text{ is chosen,} \\ 0 & \text{otherwise.} \end{cases}$$

B.4. Mathematical Model

$$\begin{aligned} \min \ & \left(\sum_{j=1}^{n} c^1_j x_j, \sum_{j=1}^{n} c^2_j x_j \right) \\ \text{s.t.} \ & \sum_{j=1}^{n} a_j x_j \le b, \\ & x_j \in \{0, 1\}, \quad j = 1, \ldots, n. \end{aligned} \tag{B1}$$

B.5. Test problems

We generate 11 different sized problems with $n = 10, 15, 20, \ldots, 60$. The number of binary variables is in the range $10 \le n \le 60$.

Appendix C. Biobjective Set-Covering Problem

The set-covering problem is a classical OR optimization problem. We have an $m \times n$ matrix A with entries 0 and 1. The problem is to determine a minimum set of columns that can cover all the rows. We generate random set-covering

problems, where we have defined a second objective that does *not cover* rows. This is easily included by adding additional slack variables. We use five times more variables than we have constraints, and we adjust the probability of a 1 in the A matrix such that, on average, 10 variables cover a row.

C.1. Indexes

m: Number of rows and slack variables.
n: Number of variables; $n = 5 \times m$.

C.2. Data

c_i^1: Cost of the columns (sets).
$a_{ij} \in \{0, 1\}$: Entries in the incidence matrix, chosen so that on average 10 columns cover each constraint.

The costs c_i^1 are generated uniformly in the interval $[1, 10]$.

C.3. Variables

$$x_j = \begin{cases} 1 & \text{if column } j \text{ is chosen,} \\ 0 & \text{otherwise;} \end{cases}$$

$$y_i = \begin{cases} 1 & \text{if row } i \text{ is not covered,} \\ 0 & \text{otherwise.} \end{cases}$$

C.4. Mathematical Model

$$\begin{aligned} \min \quad & \left(\sum_{j=1}^{n} c_j^1 x_j, \sum_{i=1}^{m} y_i \right) \\ \text{s.t.} \quad & \sum_{j=1}^{n} a_{ij} x_j + y_i \geq 1, \quad i = 1, \ldots, m; \\ & x_j \in \{0, 1\}, \quad j = 1, \ldots, n; \\ & y_i \geq 0, \quad i = 1, \ldots, m. \end{aligned} \tag{C1}$$

Note that the variables y_i, $i = 1, 2, \ldots, n$ are only required to be continuous. They will automatically be binary in all efficient solutions.

C.5. Test problems

We generate eight different sized problems with $n = 50, 100, 150, \ldots, 400$. The number of binary variables is in the range $50 \leq n \leq 400$. The number of continuous variables and the number of constraints are between $10 \leq m \leq 80$.

Appendix D. Biobjective Facility Location

Assume that m potential locations $b^i = (b_1^i, b_2^i)$, $i = 1, \ldots, m$, in the plane for locating one or more facilities with unlimited capacity are given. Establishing a facility at location b^i gives rise to a fixed cost c_i^1. Assume that these facilities will supply n potential customers with a product. The locations of the customers are $a^j = (a_1^j, a_2^j)$, $j = 1, \ldots, n$. Each customer has a demand d_j, $j = 1, \ldots, n$, which should be fulfilled. A transportation cost $c_{ij}^2 = [(b_1^i - a_1^j)^2 + (b_2^i - a_2^j)^2]^{1/2}$ from facility i to customer j is given by the Euclidean distance between the facility and the customer. The objectives in the present case are to minimize the fixed costs of establishing the facilities as well as to minimize total transportation costs.

D.1. Indexes

n: Number of customers.
m: Number of (possible) facilities.

D.2. Data

The coordinates $b^i = (b_1^i, b_2^i)$, $i = 1, \ldots, m$ of the m possible depots and the coordinates $a^j = (a_1^j, a_2^j)$, $j = 1, \ldots, n$ of the n customers are generated uniformly between 0.001 and 10. For each possible facility location i and each customer j, we define the following.

c_i^1: Cost of allocating facility i. It is calculated in this way: $10 \cdot \min(c_{ij}^2 \mid j = 1, \ldots, n)$.

$c_{i,j}^2$: Cost of transporting one unit from facility i to customer j.

d_j: Demand of customer j. It is generated uniformly in the interval $[5, 10]$.

D.3. Variables

$$x_i = \begin{cases} 1 & \text{if a facility is located in position } i, \\ 0 & \text{otherwise.} \end{cases}$$

$y_{i,j} \in [0, 1]$: The fraction of the demand of customer j that is served from facility i.

D.4. Mathematical Model

$$\begin{aligned} \min \quad & \left(\sum_{i=1}^{m} c_i^1 x_i, \sum_{i=1}^{m} \sum_{j=1}^{n} c_{i,j}^2 d_j y_{i,j} \right) \\ \text{s.t.} \quad & \sum_{i=1}^{m} y_{i,j} \geq 1, \quad j = 1, \ldots, n; \\ & y_{i,j} \leq x_i, \quad i = 1, \ldots, m, \; j = 1, \ldots, n; \\ & x_j \in \{0, 1\}, \quad j = 1, \ldots, n; \\ & y_{i,j} \in [0, 1], \quad i = 1, \ldots, m, \; j = 1, \ldots, n. \end{aligned} \tag{D1}$$

D.5. Test problems

We generate six different sized problems with $m = 5, 10, 15, 20, 30, 35$. The number of binary variables is in the range $5 \leq m \leq 35$. The number of continuous variables $n = 4 \cdot m$ is in the range $20 \leq n \leq 140$, and the number of constraints is in the range $105 \leq n + n \cdot m \leq 4{,}935$.

Appendix E. Biobjective Fixed-Charge Network Design Problem

The fixed-charge network design is a classical OR problem where a set of nodes needs to be connected, and a volume of 1 should be sent from a source node to a terminal node.

Each node is placed uniformly in a $100 * 100$ square. There are two cost functions: a fixed cost of establishing a link ($c_{\{ij\}}^1$) equal to the Euclidean distance and a cost for each unit of flow on the established links for which there is a capacity cost c_{ij}^2 also equal to the Euclidean distance. The flow is directed, but the links are undirected. If a link is established, flow can occur in both directions on the link. The requirement in the model is that there is a flow between all node pairs, from one node to the other node.

Given n nodes, there will be $n(n-1)/2$ binary variables, because these are bidirectional. The number of continuous variables is then $(n(n-1)/2) \cdot n \cdot (n-1)$.

E.1. Indexes

i, j: Nodes, used for links and flows.

k, l: Nodes, used for the demand start (k) and end nodes (l).

E.2. Data

c^1_{ij}: Fixed costs of establishing an undirected link between node i and node j, equal to the Euclidean distance between the location of nodes i and j.

c^2_{ij}: Capacity cost. This cost is oriented from i to j, but the distance is symmetric, i.e., $c^2_{ij} = c^2_{ji}$.

E.3. Variables

$$x_{ij} = \begin{cases} 1 & \text{if there is an undirected link between node } i \text{ and node } j, \\ 0 & \text{otherwise.} \end{cases}$$

y^{kl}_{ij}: The flow from i to j of the flow from source node k to terminal node l.

E.4. Mathematical Model

$$\begin{aligned} \min \quad & \left(\sum_{i=1}^{n}\sum_{j=1}^{n} c^1_{ij} x_{ij}, \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} c^2_{ij} y^{kl}_{ij} \right) \\ \text{s.t.} \quad & \sum_{j=1}^{n} y^{kl}_{ij} - \sum_{j=1}^{n} y^{kl}_{ji} = \begin{cases} 1 & \text{if } k = i, \\ -1 & \text{if } l = i, \\ 0 & \text{otherwise,} \end{cases} \qquad \forall k, l, i; \\ & y^{kl}_{ij} \le x_{ij}, \quad \forall k, l, i, j; \\ & \sum_{j=1}^{n} (x_{ij} + x_{ji}) \ge 1, \quad \forall i; \\ & x_{ij} \in \{0, 1\}, \quad \forall i, j; \\ & y^{kl}_{ij} \ge 0, \quad \forall k, l, i, j. \end{aligned} \tag{E1}$$

The last inequality in program (E1) is added to the problem to improve the LP relaxation.

E.5. Test problems

We generate nine different sized problems with $n = 3, 4, \ldots, 11$. The number of binary variables is in the range $3 \le (n \cdot (n-1))/2 \le 55$. The number of continuous variables is in the range $18 \le ((n \cdot (n-1))/2) \cdot n \cdot (n-1) \le 6{,}050$, and the number of constraints is in the range $111 \le n + n^3 + n^4 \le 15{,}983$.

Appendix F. Biobjective Random Problem

This random problem has the same number of (binary) variables and constraints.

F.1. Indexes

n: Number of rows.

n: Number of variables (columns).

F.2. Data

$c^1_j \in [1, 10]$: Cost of the first objective.

$c^2_j \in [-10, -1]$: Cost of the second objective.

$a_{ij} \in [1, 10]$: Entry in row i for variable j.

$b_i = \frac{1}{2} \cdot \sum_{j=1}^{n} a_{ij}$: Right-hand side of row i.

The numbers c^1_j, c^2_j, and a_{ij} are generated uniformly in their respective intervals.

F.3. Variables

$$x_j = \begin{cases} 1 & \text{if column } j \text{ is chosen,} \\ 0 & \text{otherwise} \end{cases}$$

F.4. Mathematical Model

$$\begin{aligned} \min \quad & \left(\sum_{j=1}^{n} c^1_j x_j, \sum_{j=1}^{n} c^2_j x_j \right) \\ \text{s.t.} \quad & \sum_{j=1}^{n} a_{ij} x_j \ge b_i, \quad i = 1, \ldots, n; \\ & x_j \in \{0, 1\}, \quad j = 1, \ldots, n \end{aligned} \tag{F1}$$

F.5. Test problems

We generate 16 different sized problems with $n = 10, 15, 20, \ldots, 80$. The number of binary variables and the number of constraints are in the range $10 \le n \le 80$.

References

Aneja YP, Nair KPK (1979) Bicriteria transportation problem. *Management Sci.* 25(1):73–78.

Bukchin J, Masin M (2004) Multi-objective design of team oriented assembly systems. *Eur. J. Oper. Res.* 156(2):326–352.

Cohon J (1978) *Multiobjective Programming and Planning* (Academic Press, New York).

Ehrgott M (2005) *Multicriteria Optimization*, 2nd ed. (Springer Verlag, Berlin).

Ehrgott M, Gandibleux X (2000) A survey and annotated bibliography of multiobjective combinatorial optimization. *OR Spektrum* 22(4):425–460.

Hooker JN (2000) *Logic-Based Methods for Optimization* (John Wiley & Sons, New York).

IBM ILOG (2012) CPLEX 12.1.1. IBM ILOG, Armonk, NY.

Kiziltan G, Yucaoglu E (1983) An algorithm for multiobjective zero-one linear programming. *Management Sci.* 29(12):1444–1453.

Klein D, Hannan E (1982) An algorithm for the multiple objective integer linear programming problem. *Eur. J. Oper. Res.* 9(4):378–385.

Land AH, Doig AG (1960) An automatic method of solving discrete programming problems. *Econometrica: J. Econometric Soc.* 28(3):497–520.

Masin M, Bukchin Y (2008) Diversity maximization approach for multiobjective optimization. *Oper. Res.* 56(2):411–424.

Mavrotas G, Diakoulaki D (1998) A branch and bound algorithm for mixed zero-one multiple objective linear programming. *Eur. J. Oper. Res.* 107(3):530–541.

Mavrotas G, Diakoulaki D (2005) Multi-criteria branch and bound: A vector maximization algorithm for mixed 0-1 multiple objective linear programming. *Appl. Math. Comput.* 171(1):53–71.

Mezmaz M, Melab N, Talbi EG (2007) A grid-based parallel approach of the multi-objective branch and bound. *15th EUROMICRO Internat. Conf. Parallel, Distributed and Network-Based Processing (PDP07)* (Institute of Electrical and Electronics Engineers, Washington, DC), 23–30.

Özpeynirci Ö, Köksalan M (2010) An exact algorithm for finding extreme supported nondominated points of multiobjective mixed integer programs. *Management Sci.* 56(12): 2302–2315.

Pedersen CR, Nielsen LR, Andersen KA (2008) The bicriterion multi-modal assignment problem: Introduction, analysis, and experimental results. *INFORMS J. Comput.* 20(3):400–411.

Przybylski A, Gandibleux X, Ehrgott M (2008) Two phase algorithms for the bi-objective assignment problem. *Eur. J. Oper. Res.* 185(2):509–533.

Przybylski A, Gandibleux X, Ehrgott M (2010a) A recursive algorithm for finding all nondominated extreme points in the outcome set of a multiobjective integer programme. *INFORMS J. Comput.* 22(3):371–386.

Przybylski A, Gandibleux X, Ehrgott M (2010b) A two phase method for multi-objective integer programming and its application to the assignment problem with three objectives. *Discrete Optim.* 7(3):149–165.

Przybylski A, Gandibleux X, Ehrgott M (2011) The two phase method for multiobjective combinatorial optimization problems. Mahjoub R, ed. *Progress in Combinatorial Optimization* (ISTE-Wiley, Hoboken, NJ), 559–596.

Raith A, Ehrgott M (2009) A two-phase algorithm for the biobjective integer minimum cost flow problem. *Comput. Oper. Res.* 36(6):1299–1331.

Sourd F, Spanjaard O (2008) A multiobjective branch-and-bound framework: Application to the biobjective spanning tree problem. *INFORMS J. Comput.* 20(3):471–484.

Sourd F, Spanjaard O, Perny P (2006) Multi-objective branch-and-bound: Application to the bi-objective spanning tree problem. *MOPGP'06: 7th Internat. Conf. Multi-Objective Programming and Goal Programming, Loire Valley, France.*

Steuer RE (1986) *Multiple Criteria Optimization: Theory, Computation and Application* (John Wiley & Sons, New York).

Tuyttens D, Teghem J, Fortemps Ph, Nieuwenhuyze VK (2000) Performance of the MOSA method for the bicriteria assignment problem. *J. Heuristics* 6(3):295–310.

Ulungu EL (1993) Optimisation combinatoire multicritère: Détermination de l'ensemble des solutions efficaces et méthodes intéractives. Ph.D. thesis, Faculté des Sciences, Université de Mons-Hainault, Mons, Belgium.

Ulungu EL, Teghem J (1995) The two phases method: An efficient procedure to solve bi-objective combinatorial optimization problems. *Foundations Comput. Decision Sci.* 20(2): 149–165.

Vincent T (2009) Multi-objective branch and bound for mixed 0-1 linear programming: Corrections and improvements. Master's thesis, Department of Computer Science, Technische Universität Kaiserslautern, Kaiserslautern, Germany.

Vincent T, Seipp F, Ruzika S, Przybylski A, Gandibleux X (2013) Multiple objective branch and bound for mixed 0-1 linear programming: Corrections and improvements for the biobjective case. *Comput. Oper. Res.* 40(1):498–509.

Visée M, Teghem J, Pirlot M, Ulungu EL (1998) Two-phases method and branch-and-bound procedures to solve the bi-objective knapsack problem. *J. Global Optim.* 12(2):139–155.

White DJ (1984) A branch and bound method for multi-objective boolean problems. *Eur. J. Oper. Res.* 15(1):126–130.