European Journal of Operational Research 268 (2018) 281–299

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Decision Support

The search-and-remove algorithm for biobjective mixed-integer linear programming problems

Banu Soylu

Department of Industrial Engineering, Erciyes University, 38039 Kayseri, Turkey

ARTICLE INFO

Article history:
Received 13 March 2017
Accepted 11 January 2018
Available online 31 January 2018

Keywords:
Multiple objective programming
Biobjective mixed-integer linear programming
Integer programming
Bound sets

ABSTRACT

In this study, an exact algorithm, called the search-and-remove (SR) algorithm, is proposed to compute the Pareto frontier of biobjective mixed-integer linear programming problems. At each stage of the algorithm, efficient slices (all integer variables are fixed in a slice) are searched with the dichotomic search algorithm and found slices are recorded and excluded from the decision space with the help of Tabu constraints. The algorithm is also enhanced with lower and upper bounds, which are updated at each stage of the algorithm. The SR algorithm continues until it is proved that all efficient slices of the biobjective mixed-integer linear programming (BOMILP) problem are found. The algorithm finally returns a set of potentially efficient slices including all efficient slices of the problem. Then, an upper envelope finding algorithm merges the Pareto frontiers of these slices to the Pareto frontier of the original problem. A computational analysis is performed on several benchmark problems and the performance of the algorithm is compared with state of the art methods from the literature.

1. Introduction

Mixed-integer linear programming problems (MILP) comprise a wide class of discrete optimization problems. In a MILP, if there are two conflicting objective functions, the problem turns into the BOMILP problem.

When all integer variables of a BOMILP problem are fixed, a biobjective linear programming problem (BOLP) called the *slice problem* (Belotti, Soylu, & Wiecek, 2013) is obtained. It is well-known in the literature that the Pareto frontier of a BOLP can easily be found with the parametric simplex algorithm (Ehrgott, 2005) or the dichotomic search algorithm (Aneja & Nair, 1979; Cohon, 1978). If any Pareto point of a slice problem contributes to the Pareto frontier of the BOMILP problem, then it is called an *efficient slice*. Since the number of efficient slices can be of exponential in the size of the input (i.e., intractable), there is no chance for a polynomial time algorithm for BOMILP. A primitive algorithm would be to solve all slice problems of the BOMILP problem, obviously this requires the complete enumeration of the set of integer variables, and then finding the upper envelope (for maximization problems) of all resulting points. The aim of the SR algorithm is precisely to improve this primitive approach in such a way as to avoid this complete enumeration.

Multiobjective MILP (MOMILP) problems are common in real life. Some examples are multiobjective hub-location (Köksalan & Soylu, 2010), multiobjective hydro-thermal self-scheduling (Ahmadi, Aghaei, Shayanfar, & Rabiee, 2012), biobjective heat and power production planning (Rong, Figueira, & Lahdelma, 2015), multiobjective energy planning (Mavrotas, Diakoulaki, & Papayannakis, 1999) etc. Many MOMILP problems exist in the literature and classical algorithms such as the ε-constraint algorithm (Haimes, Lasdon, & Wismer, 1971) have been mostly used for finding efficient solutions. However, these algorithms are only able to provide a subset of efficient solutions of MOMILP problems.

The single objective MILP literature is vast in exact and in heuristic methods, however the MOMILP algorithms literature is relatively new. Decision space search algorithms such as branch-and-bound (BB) have been applied to MOMILP as well. One of the first attempts to develop a BB algorithm for MOMILP problems is by Mavrotas and Diakoulaki (1998, 2005). Vincent, Seipp, Ruzika, Przybylski, and Gandibleux (2013) improved the BB algorithm of Mavrotas and Diakoulaki in terms of generation of the Pareto frontier. They also presented better bounds and branching strategies. Stidsen, Andersen, and Dammann (2014) proposed a biobjective BB algorithm, which can handle a subclass of biobjective mixed-binary linear programming (BOMBLP) problems, where continuous variables are only to be part of one of two objective functions. Belotti et al. (2013) presented the first general purpose BB algorithm for BOMILP problems. They also proposed improved fathoming rules to eliminate more nodes for decreasing the computation

E-mail address: bsoylu@erciyes.edu.tr

https://doi.org/10.1016/j.ejor.2018.01.026
0377-2217/

time (Belotti, Soylu, & Wiecek, 2016). Soylu and Yıldız (2015) have recently introduced a new tree based algorithm, called the local-branch-and bound algorithm, which uses the local branching concept of Fischetti and Lodi (2003).

Criterion space search algorithms have also been receiving growing attention in the literature. The first general purpose criterion space search algorithm for BOMILP problems was introduced by Boland, Charkhgard, and Savelsbergh (2015). They developed a triangle splitting algorithm, which maintains a diverse set of Pareto points throughout the algorithm. Recently, Soylu and Yıldız (2016) presented the ε, *Tabu*-constraint algorithm, which sequentially finds all Pareto line segments and points of BOMILP problems starting with one end point of the Pareto frontier.

Particularly for finding the set of extreme Pareto points of MOMILP problems, Özpeynirci and Köksalan (2010) presented an algorithm based on dichotomic search and weight space decomposition. Przybylski, Gandibleux, and Ehrgott (2010) proposed an algorithm for finding the set of extreme Pareto points of multiobjective integer linear programming (MOILP) problems.

This paper proposes an exact algorithm for BOMILP problems. The motivation is that the Pareto frontier of BOMILP problems can be computed by considering Pareto frontiers of (BOLP) slice problems and eliminating dominated points. Since BOMILP problems are in general not convex, finding all efficient slices is not easy. However, removing searched slices from the decision space makes it easier to find others. For this purpose, at each iteration efficient slices of the problem are searched with the dichotomic search algorithm. Before starting the next iteration, slices found are eliminated from the search space with the help of Tabu constraints added to the problem. These search and remove iterations repeat until it is proved that all efficient slices of the problem are found. For this purpose, upper and lower bound sets are used. Obviously, tighter bound sets lead to earlier termination of the algorithm by reducing the number of performed iterations. As a performance improvement strategy, partitioning the objective space into several subregions and parallel processing of each subregion are suggested.

In Section 2, basic definitions and models are provided. In Section 3, fundamental mechanisms of the algorithm are discussed. In Section 4, the search-and-remove iterative framework is proposed for BOMILP problems. In Section 5, computational results and performance comparisons are presented. Finally, the conclusion and further research directions are given in Section 6.

2. Basic definitions and models

Readers may refer to Isermann (1974), Yu and Zeleny (1975), Naccache (1978), Steuer (1985) and Ehrgott (2005) for a detailed coverage of multiobjective optimization.

The BOMILP problem can be formulated as follows:

$$\begin{aligned} P\text{: } & Max\ z_1\ (\tilde{\boldsymbol{x}}, \boldsymbol{x}) = \tilde{\boldsymbol{c}}_1^T\tilde{\boldsymbol{x}} + \boldsymbol{c}_1^T\boldsymbol{x} \\ & Max\ z_2(\tilde{\boldsymbol{x}}, \boldsymbol{x}) = \tilde{\boldsymbol{c}}_2^T\tilde{\boldsymbol{x}} + \boldsymbol{c}_2^T\boldsymbol{x} \\ & Subject\ to\ (\tilde{\boldsymbol{x}}, \boldsymbol{x}) \in X \end{aligned}$$

where the set $X := \{\ (\tilde{\boldsymbol{x}}, \boldsymbol{x}) \in \mathbb{R}_+^p \times \mathbb{Z}_+^{n-p} :\ \tilde{A}\tilde{\boldsymbol{x}} + A\boldsymbol{x} \leqq \boldsymbol{b}\ \}$ is the set of feasible solutions defined in the decision space. The vectors $\tilde{\boldsymbol{c}}_1,\ \tilde{\boldsymbol{c}}_2 \in \mathbb{R}^p$ and $\boldsymbol{c}_1,\ \boldsymbol{c}_2 \in \mathbb{R}^{n-p}$ are cost vectors, the matrices $\tilde{A} \in \mathbb{R}^{m\times p}, A \in \mathbb{R}^{m\times\ (n-p)}$ are coefficient matrices of m constraints, and $\boldsymbol{b} \in \mathbb{R}^m$ is the right-hand side vector. There are p continuous and $n-p$ integer variables. It is assumed that each integer variable x_i is bounded, i.e. $0 \le l_i \le x_i \le u_i < +\infty$ for $i = 1, 2, \ldots\ldots, n-p$ where l_i and u_i are lower and upper bound values, respectively. It is also assumed that each continuous variable $\tilde{x}_i$ is bounded, i.e. $0 \le \tilde{x}_i \le u_i < +\infty$ for $i = 1, 2, \ldots\ldots, p$. Here, the biobjective integer programming (BOIP) problem, where $p = 0$, and the biobjective linear programming (BOLP) problem, where $p = n$, are considered as special classes of the BOMILP problem. The image of the set X is $Y := z(X) := \{\boldsymbol{y} \in \mathbb{R}^2 :\ \boldsymbol{y} = z(\tilde{\boldsymbol{x}}, \boldsymbol{x})$ for some $(\tilde{\boldsymbol{x}}, \boldsymbol{x}) \in X\ \}$ called the set of attainable vectors defined in the objective/criterion space $\mathbb{R}^2$. Here $\boldsymbol{y} = z(\tilde{\boldsymbol{x}}, \boldsymbol{x})$ where $z = (z_1, z_2)$.

Definition 1. A feasible solution $(\tilde{\boldsymbol{x}}^*, \boldsymbol{x}^*) \in X$ is said to be *efficient* for problem P if and only if there is no $(\tilde{\boldsymbol{x}}, \boldsymbol{x}) \in X$ such that $z(\tilde{\boldsymbol{x}}, \boldsymbol{x}) \ge z(\tilde{\boldsymbol{x}}^*, \boldsymbol{x}^*)$. If $(\tilde{\boldsymbol{x}}^*, \boldsymbol{x}^*) \in X$ is efficient, then $z(\tilde{\boldsymbol{x}}^*, \boldsymbol{x}^*)$ is called a *Pareto point*. If $(\tilde{\boldsymbol{x}}^1, \boldsymbol{x}^1)$, $(\tilde{\boldsymbol{x}}^2, \boldsymbol{x}^2) \in X$ and $z(\tilde{\boldsymbol{x}}^1, \boldsymbol{x}^1) \ge z(\tilde{\boldsymbol{x}}^2, \boldsymbol{x}^2)$, then it is said that $(\tilde{\boldsymbol{x}}^1, \boldsymbol{x}^1)$ dominates $(\tilde{\boldsymbol{x}}^2, \boldsymbol{x}^2)$, and $z(\tilde{\boldsymbol{x}}^1, \boldsymbol{x}^1)$ dominates $z(\tilde{\boldsymbol{x}}^2, \boldsymbol{x}^2)$. The set of all efficient solutions $(\tilde{\boldsymbol{x}}^*, \boldsymbol{x}^*) \in X$ is denoted by X_E. The set of all Pareto points $z(\tilde{\boldsymbol{x}}^*, \boldsymbol{x}^*) \in Y$ for some $(\tilde{\boldsymbol{x}}^*, \boldsymbol{x}^*) \in X_E$ is denoted by Y_N, also referred to as the *Pareto frontier*.

Given two vectors $\boldsymbol{y}^1,\ \boldsymbol{y}^2 \in \mathbb{R}^2$, the following notation is used:

$\boldsymbol{y}^1 \geqq \boldsymbol{y}^2$ if $y_k^1 \ge y_k^2$ for all $k = 1, 2$
$\boldsymbol{y}^1 \ge \boldsymbol{y}^2$ if $\boldsymbol{y}^1 \geqq \boldsymbol{y}^2$ and $\boldsymbol{y}^1 \ne \boldsymbol{y}^2$
$\boldsymbol{y}^1 > \boldsymbol{y}^2$ if $y_k^1 > y_k^2$ for all $k = 1, 2$.

The set $\mathbb{R}^2_{\geqq}$ is defined as $\mathbb{R}^2_{\geqq} := \{\boldsymbol{y} \in \mathbb{R}^2 : \boldsymbol{y} \geqq 0\}$, and sets $\mathbb{R}^2_{>}$, $\mathbb{R}^2_{\ge}$, $\mathbb{R}^2_{<}$, $\mathbb{R}^2_{\le}$, $\mathbb{R}^2_{\leqq}$ are defined similarly. Let S be an arbitrary set. Then, the set $S^{\geqq}$ is defined as $S^{\geqq} = S + \mathbb{R}^2_{\geqq} := \{s + \boldsymbol{y} : s \in S, \boldsymbol{y} \in \mathbb{R}^2_{\geqq}\}$, and sets $S^{>}$, $S^{\ge}$, $S^{<}$, $S^{\le}$, $S^{\leqq}$ are defined similarly.

Definition 2. A set $S \subset \mathbb{R}^2$ is called *connected* if there are no open sets O_1, O_2 such that $S \subset O_1 \cup O_2$, $S \cap O_1 \ne \emptyset$, $S \cap O_2 \ne \emptyset$, $S \cap O_1 \cap O_2 = \emptyset$.

Definition 3. A feasible solution $(\tilde{\boldsymbol{x}}', \boldsymbol{x}') \in X$ is said to be *weakly efficient* for problem P if and only if there is no $(\tilde{\boldsymbol{x}}, \boldsymbol{x}) \in X$ such that $z(\tilde{\boldsymbol{x}}, \boldsymbol{x}) > z(\tilde{\boldsymbol{x}}', \boldsymbol{x}')$. Then the point $\boldsymbol{y}' = z(\tilde{\boldsymbol{x}}', \boldsymbol{x}')$ is called a *weak Pareto point*.

Definition 4. Let $(\tilde{\boldsymbol{x}}^a, \boldsymbol{x}^a) \in X_E$. If there is some $\lambda \in (0, 1)$ such that $(\tilde{\boldsymbol{x}}^a, \boldsymbol{x}^a) \in X_E$ is an optimal solution of $max_{(\tilde{\boldsymbol{x}}, \boldsymbol{x}) \in X}\ \{\lambda z_1(\tilde{\boldsymbol{x}}, \boldsymbol{x}) + (1-\lambda) z_2(\tilde{\boldsymbol{x}}, \boldsymbol{x})\}$, then $(\tilde{\boldsymbol{x}}^a, \boldsymbol{x}^a) \in X_E$ is called a *supported efficient solution* and $\boldsymbol{y}^a = z(\tilde{\boldsymbol{x}}^a, \boldsymbol{x}^a)$ is called a *supported Pareto point*. Otherwise, $(\tilde{\boldsymbol{x}}^a, \boldsymbol{x}^a) \in X_E$ is called a *nonsupported efficient solution* and $\boldsymbol{y}^a = z(\tilde{\boldsymbol{x}}^a, \boldsymbol{x}^a)$ is called a *nonsupported Pareto point*.

Definition 5. A Pareto point $\boldsymbol{y}^a = z(\tilde{\boldsymbol{x}}^a, \boldsymbol{x}^a) \in Y_N$ is called an *extreme supported Pareto (ExSP) point* if $\boldsymbol{y}^a$ is an extreme point of $Conv(Y)$. The set Y_{XN} denotes the set of *ExSP* points.

Definition 6. Let $Y_{XN} = \{\boldsymbol{y}^1, \boldsymbol{y}^2, \ldots, \boldsymbol{y}^h\}$ such that $y_1^1 < y_1^2 < \ldots < y_1^h$. Points $\boldsymbol{y}^j$ and $\boldsymbol{y}^{j+1}$, $\forall j = 1, 2, \ldots, h-1$ are called *adjacent*.

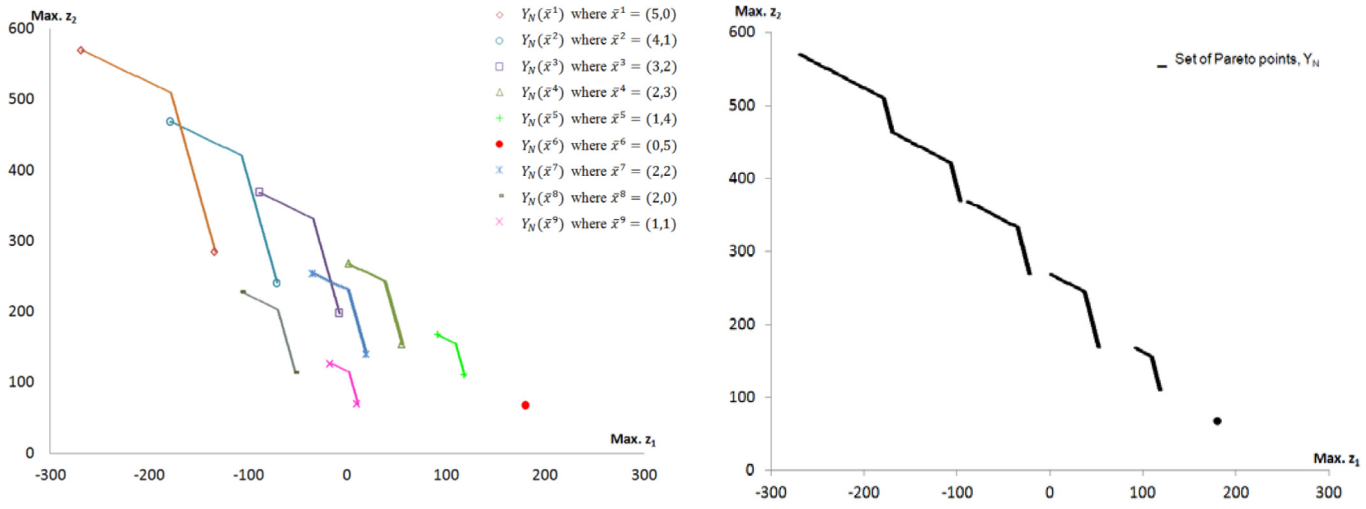
Definition 7. Let $\boldsymbol{y}^a,\ \boldsymbol{y}^b \in Y$ be two attainable points such that $y_1^a < y_1^b$ and $y_2^a > y_2^b$. The point $\check{\boldsymbol{y}} = (y_1^a, y_2^b)$ is called the *local nadir point* with respect to $\boldsymbol{y}^a$ and $\boldsymbol{y}^b$. The point $\check{\check{\boldsymbol{y}}} = (\check{\check{y}}_1, \check{\check{y}}_2)$ given by $\check{\check{y}}_k = \min_{\boldsymbol{y} \in Y_N} y_k$ for $k = 1, 2$ is called the *nadir point*. The point $\boldsymbol{y}^I = (y_1^I, y_2^I)$ given by $y_k^I = max\{z_k(\tilde{\boldsymbol{x}}, \boldsymbol{x}) : (\tilde{\boldsymbol{x}}, \boldsymbol{x}) \in X\}$ for $k = 1, 2$ is called the *ideal point*.

Given a weight $\lambda \in (0, 1)$ the single objective weighted-sum problem P_λ is defined as $max_{(\tilde{\boldsymbol{x}}, \boldsymbol{x}) \in X}\ \{\lambda z_1(\tilde{\boldsymbol{x}}, \boldsymbol{x}) + (1-\lambda) z_2(\tilde{\boldsymbol{x}}, \boldsymbol{x})\}$.

A *slice problem* of P is a BOLP obtained by fixing the integer variables of P and defined as follows (Belotti et al., 2013):

$$\begin{aligned} P\left(\tilde{\boldsymbol{x}}^j\right)\text{: } & Max\ z_1(\tilde{\boldsymbol{x}}) = \tilde{\boldsymbol{c}}_1^T\tilde{\boldsymbol{x}} + \boldsymbol{c}_1^T\tilde{\boldsymbol{x}}^j \\ & Max\ z_2(\tilde{\boldsymbol{x}}) = \tilde{\boldsymbol{c}}_2^T\tilde{\boldsymbol{x}} + \boldsymbol{c}_2^T\tilde{\boldsymbol{x}}^j \\ & Subject\ to\ \tilde{\boldsymbol{x}} \in X\left(\tilde{\boldsymbol{x}}^j\right) \end{aligned}$$

where $X(\tilde{\boldsymbol{x}}^j) := \{\ \tilde{\boldsymbol{x}} \in \mathbb{R}_+^p :\ \tilde{A}\tilde{\boldsymbol{x}} \leqq \boldsymbol{b} - A\tilde{\boldsymbol{x}}^j$ for a given $\tilde{\boldsymbol{x}}^j \in \mathbb{Z}_+^{n-p}\}$ defines a slice of the set X. Here the $\tilde{\boldsymbol{x}}^j$ vector refers to integer values defined priorily. The set $Y(\tilde{\boldsymbol{x}}^j) := z(X(\tilde{\boldsymbol{x}}^j)) :=$



(a) Pareto frontiers of several slices (b) Pareto frontier of the example BOMILP

Fig. 1. An example BOMILP problem.

$\{\boldsymbol{y} \in \mathbb{R}^2 : \ \boldsymbol{y} = z(\tilde{\boldsymbol{x}}) \ \text{ for some } \tilde{\boldsymbol{x}} \in X(\bar{\boldsymbol{x}}^j) \ \}$ is the set of attainable vectors of problem $P(\bar{\boldsymbol{x}}^j)$. Let $X_E(\bar{\boldsymbol{x}}^j)$ and $Y_N(\bar{\boldsymbol{x}}^j)$ denote the set of efficient solutions and Pareto points of problem $P(\bar{\boldsymbol{x}}^j)$, respectively. If a point $\boldsymbol{y} \in Y_N(\bar{\boldsymbol{x}}^j)$ is also in the set of Pareto points of P, i.e. $\boldsymbol{y} \in Y_N$, then the slice $X(\bar{\boldsymbol{x}}^j)$ is called an efficient slice.

It is well-known that the set of Pareto points of BOLP is connected and all Pareto points are supported (Naccache, 1978; Yu & Zeleny, 1975).

Definition 8. Let $\boldsymbol{y}^a$ and $\boldsymbol{y}^b$ be such that $y_1^a < y_1^b$ and $y_2^a > y_2^b$, then $[\boldsymbol{y}^a, \boldsymbol{y}^b]$ denotes a closed line segment. A line segment may be open $()$, half open $(],[)$ or closed $[]$ depending on whether the end points are dominated or not. If all points on the line segment are Pareto, then it is called a *Pareto line segment*. The $Y_N(\bar{\boldsymbol{x}}^j)$ set is characterized by finitely many Pareto line segments, which are defined by adjacent *ExSP* points. Any point $\boldsymbol{y}$ of a line segment $[\boldsymbol{y}^a, \boldsymbol{y}^b]$ can be generated by taking the convex combination of end points as follows: $\boldsymbol{y} = \lambda \boldsymbol{y}^a + (1-\lambda)\boldsymbol{y}^b$ where $0 \le \lambda \le 1$.

Example 1. Let us consider the following parameters for problem P (Belotti et al., 2013).

$$\tilde{\boldsymbol{c}}_1^T = [-3 \quad -6 \quad 3 \quad 5] \qquad \boldsymbol{c}_1^T = [0 \quad 0]$$
$$\tilde{\boldsymbol{c}}_2^T = [15 \quad 4 \quad 1 \quad 2] \qquad \boldsymbol{c}_2^T = [0 \quad 0]$$

$$\tilde{A} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} A = \begin{bmatrix} 3 & 0 \\ -6 & 0 \\ 3 & 0 \\ -6 & 0 \\ 0 & 4 \\ 0 & -4.5 \\ 0 & 4 \\ 0 & -4.5 \\ 1 & 1 \end{bmatrix} \mathrm{b} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 5 \end{bmatrix}$$

Where $(\tilde{\boldsymbol{x}}, \boldsymbol{x}) \in \mathbb{R}_+^4 \times \mathbb{Z}_+^2$.

Fig. 1(a) shows Pareto points of some slices for this example. Here a slice is identified with the integer valued vector $\bar{\boldsymbol{x}}^j$. It can be seen that the set of Pareto points of each slice is connected and all Pareto points are supported. Observe that $Y_N(\bar{\boldsymbol{x}}^6)$ is an isolated point. Pareto points of six slices, i.e. slices $X(\bar{\boldsymbol{x}}^1), X(\bar{\boldsymbol{x}}^2), X(\bar{\boldsymbol{x}}^3), X(\bar{\boldsymbol{x}}^4), X(\bar{\boldsymbol{x}}^5)$ and $X(\bar{\boldsymbol{x}}^6)$, contribute to the Pareto frontier of P, so they are also *efficient slices* of the problem. Fig. 1(b) shows the Pareto frontier of the example BOMILP problem. It is composed of some Pareto line segments and an isolated Pareto point. Although all Pareto points of each slice are supported, there are nonsupported Pareto points in the BOMILP problem.

2.1. Subproblem with Tabu constraints

Before defining the *Tabu constraint*, definition of the *Hamming distance* is presented. The *Hamming distance* (Hamming, 1950) is a natural similarity measure between two integer valued vectors. Accordingly, the *Hamming distance* between the integer part of a solution vector, i.e. $\boldsymbol{x} \in \mathbb{Z}_+^{n-p}$, and the integer valued vector $\bar{\boldsymbol{x}}^j \in \mathbb{Z}_+^{n-p}$ can be computed as $H(\boldsymbol{x}, \bar{\boldsymbol{x}}^j) = \sum_{i=1}^{n-p} |x_i - \bar{x}_i^j|$. Similarly, the distance between any two solutions is defined by the Hamming distance of their integer parts. For the example given in Fig. 1, the Hamming distance between $\bar{\boldsymbol{x}}^1$ and $\bar{\boldsymbol{x}}^2$ is $H(\bar{\boldsymbol{x}}^1, \bar{\boldsymbol{x}}^2) = 2$. The *Tabu constraint* (Fischetti & Lodi, 2003), which is also known as the *no-good constraint* (Hooker, 2011), is defined as $H(\boldsymbol{x}, \bar{\boldsymbol{x}}^j) \ge 1$. It is first introduced to the BOMILP field by Stidsen et al. (2014) for binary variables. This constraint restricts the feasible set and avoids finding a solution belonging to slice $X(\bar{\boldsymbol{x}}^j)$. Therefore, it excludes a found slice $X(\bar{\boldsymbol{x}}^j)$ from the feasible set. Although it is nonlinear, it can be linearized as stated by Fischetti, Glover, and Lodi (2005):

$$H\left(\boldsymbol{x}, \bar{\boldsymbol{x}}^j\right) = \sum_{\substack{i=1 \\ \bar{x}_i^j = 0}}^{n-p} x_i \ + \ \sum_{\substack{i=1 \\ \bar{x}_i^j = 1}}^{n-p} (1 - x_i) \tag{1}$$

Eq. (1) is valid for binary vectors only. For integer valued variables, the linearization is a little more complex. In this case, we need to consider upper (u_i) and lower (l_i) bounds on the value of x_i, i.e. $l_i \le x_i \le u_i$, $\forall i = 1, 2, \ldots\ldots, n-p$ as follows.

$$H\left(\boldsymbol{x}, \bar{\boldsymbol{x}}^j\right) = \sum_{\substack{i=1 \\ \bar{x}_i^j = l_i}}^{n-p} (x_i - l_i) + \sum_{\substack{i=1 \\ \bar{x}_i^j = u_i}}^{n-p} (u_i - x_i) + \sum_{\substack{i=1 \\ l_i < \ \bar{x}_i^j < u_i}}^{n-p} \left(x_i^+ + x_i^-\right) \tag{2}$$

where $x_i - x_i^+ + x_i^- = \bar{x}_i^j$, and $x_i^+ x_i^- = 0$, $x_i^+, \ x_i^- \ge 0 \ \forall i : \ l_i < \ \bar{x}_i^j < u_i$

In Eq. (2), slack and surplus variables should not be positive at the same time. Therefore, the constraint set $x_i^+ x_i^- = 0$ is used. It is also nonlinear but can easily be linearized, however, the linearization requires defining an additional binary variable and two

constraints for each integer variable x_i, $\forall i: \; l_i < \; \tilde{x}^j_{i-p} < u_i$ as follows:

$$\begin{aligned} &x_i^+ \leq M bin_i \\ &x_i^- \leq M(1 - bin_i) \\ &bin_i \in \{0, 1\} \end{aligned} \tag{3}$$

where M can be chosen as any number greater than u_i.

Example 2. For the Pareto frontier of slice $X(\tilde{\boldsymbol{x}}^6)$ given in Fig. 1, the Hamming distance from any solution $(\tilde{\boldsymbol{x}}, \boldsymbol{x})$ can be written as follows.

$$H\left(\boldsymbol{x}, \tilde{\boldsymbol{x}}^6\right) = |x_5 - 0| + |x_6 - 5| \tag{4}$$

After applying the above linearization into the Tabu constraint $H(\boldsymbol{x}, \tilde{\boldsymbol{x}}^6) \geq 1$, the following set of constraints is obtained:

$$(x_5 - 0) + \left(x_6^+ + x_6^-\right) \geq 1 \tag{5}$$

$$x_6 - x_6^+ + x_6^- = 5 \tag{6}$$

$$x_6^+ \leq M bin_6 \tag{7}$$

$$\begin{aligned} &x_6^- \leq M(1 - bin_6) \\ &bin_6 \in \{0, 1\} \end{aligned} \tag{8}$$

If problem P is augmented with Tabu constraints, then it is called a *subproblem*. If this augmentation is realized at iteration t of the algorithm, then it is called the *subproblem* P_t. Note that $P_1 = P$. Actually, the subproblem P_t is derived from the subproblem P_{t-1} and the slices identified in iteration $t-1$.

Example 3. Assume that slices $X(\tilde{\boldsymbol{x}}^1)$ and $X(\tilde{\boldsymbol{x}}^6)$ are explored at iteration $t-1$. Then the subproblem $P_t = P_{t-1} \cap \{H(\boldsymbol{x}, \tilde{\boldsymbol{x}}^1) \geq 1, \; H(\boldsymbol{x}, \tilde{\boldsymbol{x}}^6) \geq 1\}$.

The set of feasible solutions of P_t is X_t and the set of attainable points of P_t is Y_t. The sets X_{tE}, Y_{tN}, and Y_{tXN} denote the set of efficient solutions, Pareto points, and extreme supported Pareto points of problem P_t, respectively. Problem $P_{\lambda,t}$ refers to the single objective weighted-sum problem of P_t. Additionally, $\tilde{P}$, $\tilde{P}_t$ and $\tilde{P}(\tilde{\boldsymbol{x}}^j)$ refer to LP relaxations of corresponding problems.

3. Fundamental mechanisms

In this section, details of important mechanisms of the SR algorithm such as dichotomic search for BOMILP problems, bound sets, merging the Pareto frontier are presented.

The search-and-remove mechanism of the algorithm is very simple and different from existing algorithms in the literature. The search mechanism is performed with the dichotomic search algorithm, while the remove mechanism is performed with Tabu constraints. In order to avoid searching the feasible set X and to stop the algorithm, bound sets are used. Finally, in order to merge the Pareto frontier an upper envelope finding algorithm is utilized.

3.1. Dichotomic search for BOMILP problems

The dichotomic search (DS) algorithm (Aneja & Nair, 1979; Cohon, 1978) computes *ExSP* points of a BOIP/BOLP/BOMILP. The main idea involves taking two *ExSP* points, calculating a weight vector perpendicular to the line passing through these two *ExSP* points and solving the weighted-sum problem with the computed weight vector. If there is a new solution improving the objective function, then two new weight vectors are generated. Otherwise, this search direction is skipped and concluded that there is no other *ExSP* point in this direction. Whenever all directions are searched, the algorithm returns the set of *ExSP* points.

Here our aim is to determine each *ExSP* point of P_t and its associated slice. These slices are also the potentially efficient slices of problem P, i.e. Pareto line segments of these slices might contribute to the Pareto frontier of problem P. Additionally, Tabu constraints of the next iteration are defined by these slices. Fig. 2 illustrates the DS principle on a subproblem P_t (actually P_1) of BOMILP example given in Fig. 1. First, two extreme points, $\boldsymbol{y}^a$ and $\boldsymbol{y}^b$, are found by using lexicographic optimization as follows (Fig. 2(a)): $\boldsymbol{y}^b = z(\tilde{\boldsymbol{x}}^b, \boldsymbol{x}^b) = lexmax_{(\tilde{\boldsymbol{x}}, \boldsymbol{x}) \in \boldsymbol{X}} \; \{z_1(\tilde{\boldsymbol{x}}, \boldsymbol{x}), \; z_2(\tilde{\boldsymbol{x}}, \boldsymbol{x})\}$ and $\boldsymbol{y}^a = z(\tilde{\boldsymbol{x}}^a, \boldsymbol{x}^a) = lexmax_{(\tilde{\boldsymbol{x}}, \boldsymbol{x}) \in \boldsymbol{X}} \; \{z_2(\tilde{\boldsymbol{x}}, \boldsymbol{x}), \; z_1(\tilde{\boldsymbol{x}}, \boldsymbol{x})\}$, where *lexmax* implies the ranking of objective functions such that the first mentioned objective function has the highest priority, and only if alternative optimal solutions exist, the other objective function can be considered. Then, a search direction λ is computed (Fig. 2(b)). When we solve the weighted-sum problem P_λ over this direction, we have found a new solution and so two new search directions, λ' and λ'' are obtained. The objective function does not improve over the λ' direction (Fig. 2(c)). That means there is no *ExSP* point in this direction. Therefore, we can stop the search in this direction. The objective function does also not improve in the λ'' direction (Fig. 2(d)). Therefore, we can stop the search in this direction as well. Finally, the DS algorithm returns three *ExSP* points belonging to two efficient slices, i.e. slices $X(\tilde{\boldsymbol{x}}^1)$ and $X(\tilde{\boldsymbol{x}}^6)$. By applying the parametric simplex algorithm (or the DS algorithm) to each slice problem separately, we can obtain their Pareto frontiers (Fig. 2(e)).

Proposition 1. *If all ExSP points of problem P belong to a single slice, then the problem* P *and the slice problem have the same set of Pareto points.*

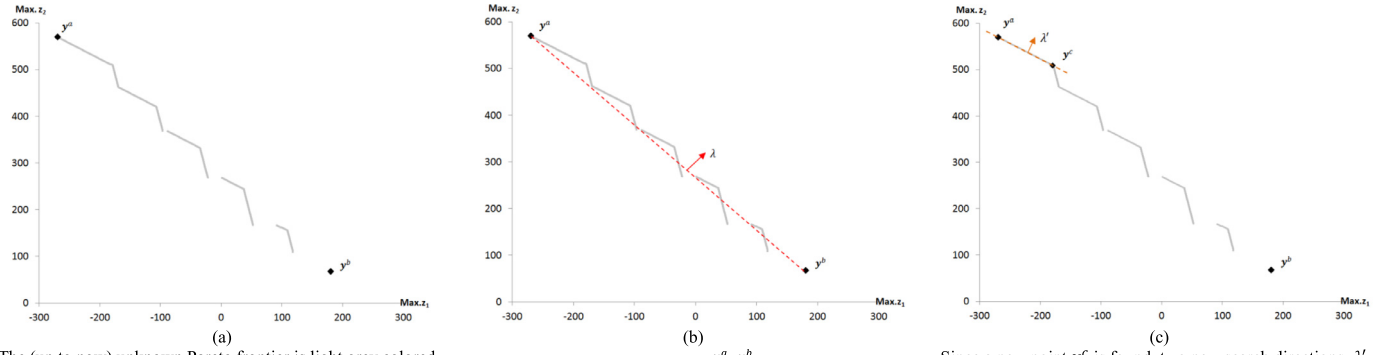
Proof. Pareto points of a slice problem are connected and supported. If all *ExSP* points of problem P belong to a single slice, then all points of other slices are dominated by or equal to the Pareto points of that particular slice. Else, there would be an *ExSP* point from another slice. □

3.2. The parametric simplex algorithm for BOLP problems

The details of the Parametric Simplex Algorithm (PSA) for BOLPs are explained in Ehrgott (2005). Briefly, in every iteration of the PSA a new optimal basis, which defines a basic feasible solution of the parametric linear program, is found. PSA moves from one vertex of the feasible region to another until it finds the set of all *ExSP* points. Whenever the adjacent *ExSP* points are connected by a Pareto line segment, the Pareto frontier of the slice problem is obtained. Simplex-type algorithms are known to work better if the problem is not degenerate (Rudloff, Ulus, & Vanderbei, 2017). In this study, the PSA is used in order to find the Pareto frontier of BOLPs.

3.3. Upper and lower bound sets

Bounds help to discard subregions of the feasible region. The notion of limiting the feasible region to be searched by using bound sets in dynamic programming is presented by Villarreal and Karwan (1981). Rong and Figueira (2014) used bound sets in a dynamic programming algorithm for the biobjective knapsack problem. Ehrgott and Gandibleux (2007) present bound sets for biobjective combinatorial optimization problems. Recently, bound sets based on the convex hull of the Pareto points of the problem have been presented by Delort and Spanjaard (2010), Ehrgott and Gandibleux (2007). Bound sets used in a biobjective BB framework are by Gadegaard, Ehrgott, and Nielsen (2016), Sourd and Spanjaard (2008) and Vincent et al. (2013).



(a)

The (up to now) unknown Pareto frontier is light grey colored. First, two extreme Points, $\boldsymbol{y}^a$ and $\boldsymbol{y}^b$, are computed.

(b)

The new search direction $\lambda = \frac{y_2^a - y_2^b}{y_2^a - y_2^b + y_1^b - y_1^a}$ is perpendicular to the line passing through these two points.

(c)

Since a new point $\boldsymbol{y}^c$ is found, two new search directions, λ' and λ'', are obtained. There is no *ExSP* point improving the objective function over the λ' direction. $\lambda' = \frac{y_2^a - y_2^c}{y_2^a - y_2^c + y_1^c - y_1^a}$

(d)

There is no ExSP point improving the objective function over the λ'' direction. So, the DS algorithm returns three *ExSP* points.

(e)

After applying PSA to each slice problem, we get their Pareto frontier as well as partial Pareto frontier of problem *P*.

Fig. 2. Iterations of the dichotomic search principle over an example BOMILP.

In this study, several bound sets are computed at each iteration of the SR algorithm. Bound sets also determine the stopping condition of the SR algorithm. Whenever it is proven that no more Pareto points can be found by searching the subproblem P_t, the SR algorithm stops.

A lower bound (LB) set is related with the potentially Pareto points, which are Pareto points of *P* but unknown yet, implied by the current set of attainable points of the problem *P*. Let $\hat{Y}$ be the set of attainable points of *P* that have been found so far and let $\breve{\theta}$ be the set of local nadir points implied by each adjacent pair of points in $\hat{Y}$. The $\hat{Y}$ set is the lower bound set in this study. The potentially Pareto points are located in the set $\breve{\theta}^{>}$. If the set $\hat{Y} = Y_N$, then we would have the tightest lower bound set.

Example 4. Referring to four *ExSP* points (associated with slices $X(\bar{\boldsymbol{x}}^1)$ and $X(\bar{\boldsymbol{x}}^6)$) given in Fig. 2(e), Fig. 3(a) illustrates the lower bound set and the set $\breve{\theta}^{>}$ encountered at the first iteration of the SR algorithm for the example problem. Here the set $\hat{Y}$ only includes four attainable points available up to now. We update the set $\hat{Y}$ with the *ExSP* points of a subproblem at each iteration. Furthermore, if the set $\hat{Y}$ is updated with feasible line segments found, this will help to make tighter the LB set as in Fig. 3(b).

An upper bound (UB) set dominates all Pareto points of P_t. A well-known upper bound set for the subproblem P_t is the ideal point y^{tI} of P_t. However, it is weak since the ideal point y^{tI} is typically far away from the Pareto frontier of P_t. Through the convex hull of *ExSP* points of P_t or $\tilde{P}_t$, a valid upper bound set can be defined. Thus, we have the following upper bound sets.

i. $\tilde{y}^{tI}$
ii. y^{tI}
iii. $\tilde{Y}_{tN}$
iv. $conv(Y_{tXN})_N$

If the set Y_{tN} was known, then we would have the tightest UB set. Above, (ii) is tighter than (i), while (iv) is tighter than (iii). Since computing (i) and (iii) is easier, the proposed algorithm first applies them. If the stopping condition is not satisfied, then it computes (ii) and (iv).

Example 5. At the second iteration (i.e. the subproblem $P_2 = P_1 \cap \{H(\boldsymbol{x}, \bar{\boldsymbol{x}}^1) \geq 1,\ H(\boldsymbol{x}, \bar{\boldsymbol{x}}^6) \geq 1\}$ is now considered), the DS algorithm finds some solutions of three new slices, i.e. $X(\bar{\boldsymbol{x}}^2)$, $X(\bar{\boldsymbol{x}}^5)$ and $X(\bar{\boldsymbol{x}}^7)$. Fig. 4(a) illustrates Pareto frontiers of these slices. Fig. 4(b) represents the Pareto frontier of the relaxed subproblem $\tilde{P}_2$, i.e. $\tilde{Y}_{2N}$, and the ideal point $\tilde{y}^{2I}$. The proposed algorithm first uses upper bound sets obtained through the LP relaxed subproblem as in Fig. 4(b). If they are unable to satisfy the stopping condition of the algorithm, it then uses upper bound sets obtained through the MILP subproblem, i.e. the ideal point y^{tI} and the convex hull of *ExSP* points of P_t (i.e. $conv(Y_{tXN})_N$), as presented in Fig. 4(c).

Proposition 2. *If one of the following conditions is realized, then the subproblem P_t does not include any Pareto point $y \in Y_N$, and the SR algorithm stops.*

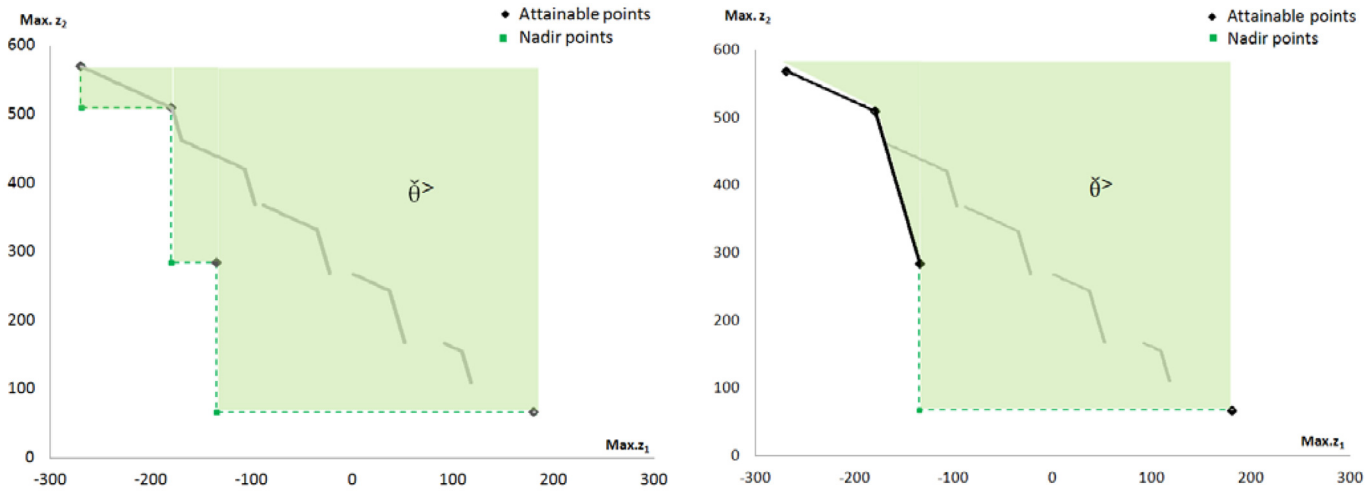
$$\text{Condition}\quad 1 : \tilde{y}^{tI\leqq} \cap \breve{\theta}^{>} = \emptyset, \tag{9}$$

$$\text{Condition}\quad 2 : y^{tI\leqq} \cap \breve{\theta}^{>} = \emptyset, \tag{10}$$

$$\text{Condition}\quad 3 : \tilde{Y}_{tN}^{\leqq} \cap \breve{\theta}^{>} = \emptyset, \tag{11}$$

$$\text{Condition}\quad 4 : conv(Y_{tXN})_N^{\leqq} \cap \breve{\theta}^{>} = \emptyset \tag{12}$$

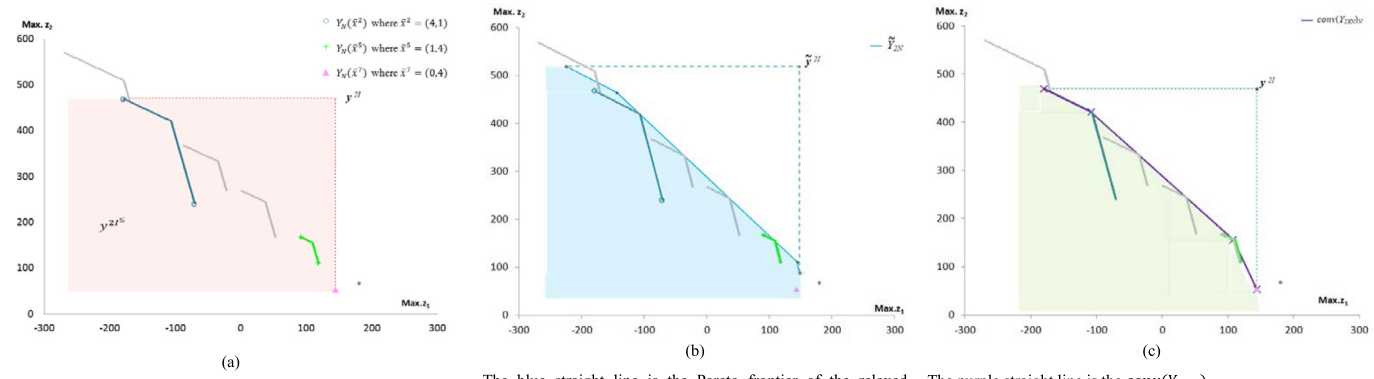
Proof. Let us first consider *condition* 1. It is clear that $\tilde{y}^{tI\leqq} \supseteq Y_{tN}$ and $\breve{\theta}^{>} \supseteq Y_N$. Hence the condition $\tilde{y}^{tI\leqq} \cap \breve{\theta}^{>} = \emptyset$ implies that



The (up to now) unknown Pareto frontier is light grey colored. Four attainable points are from Figure 2(e). The shaded area represents the set $\breve{\theta}^{>}$, which contains potentially Pareto points of P.

If line segments were inserted, the shaded area would be smaller.

Fig. 3. Lower bound sets encountered at the first iteration of the algorithm (a) a weaker one requires less computational effort (b) a tighter one requires more computational effort.



After eliminating slices found at the first iteration, three new slices, i.e. $X(\overline{x}^2)$, $X(\overline{x}^5)$ and $X(\overline{x}^7)$, are obtained by the DS. y^{2l} is the ideal point of P_2. The shaded area is the set $y^{2l\leqq}$.

The blue straight line is the Pareto frontier of the relaxed subproblem $\tilde{P}_2$, while $\tilde{y}^{2l}$ is the ideal point of it.

The purple straight line is the $conv(Y_{2XN})_N$

Fig. 4. Upper bound sets encountered at the second iteration of the algorithm (a) ideal point of the subproblem (b) upper bound sets over the LP relaxed subproblem (c) upper bound sets over the *ExSP* points of the subproblem.

$Y_{tN} \cap Y_N = \emptyset$. That means there is no need to further search the subproblem P_t since it does not include any point $\boldsymbol{y} \in Y_N$. A similar proof can also be done for the other conditions. □

In practice, it is possible to check these conditions as follows:

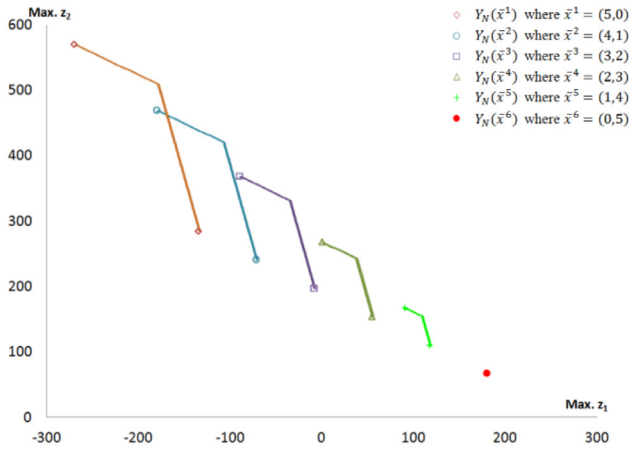
If the ideal point $\tilde{y}^{tI}$ of subproblem $\tilde{P}_t$ is dominated by any attainable point $\hat{\boldsymbol{y}} \in \hat{Y}$, then condition 1 is satisfied and there is no need to further search subproblem P_t. This is also one of the classical fathoming rules used in multiobjective BB algorithms (Bitran & Rivera, 1982; Mavrotas and Diakoulaki, 1998; Visée, Teghem, Pirlot, & Ulungu, 1998). Since the ideal point is generally far away from the Pareto frontier, this condition is usually ineffective.

Similar to the above case, if the ideal point y^{tI} of subproblem P_t is dominated by any attainable point $\hat{\boldsymbol{y}} \in \hat{Y}$, then condition 2 is satisfied and hence there is no need to further search subproblem P_t.

Condition 3 can be checked with the help of another fathoming rule by Belotti et al. (2016), Ehrgott and Gandibleux (2007) and Visée et al. (1998). For a given $\lambda \in [0, 1]$, if the worst parametric objective function value in $\breve{\theta}$ (*i.e.* $\min_{\breve{y}\in\breve{\theta}}\{\lambda\breve{y}_1 + (1-\lambda)\breve{y}_2\}$), is greater than or equal to the best parametric objective function value of subproblem $\tilde{P}_t$ (i.e. $\max_{\boldsymbol{y}\in\tilde{Y}_{tXN}}\{\lambda y_1 + (1-\lambda)y_2\}$, then condition 3 is satisfied. Note that subproblem $\tilde{P}_t$ admits the maximum parametric objective function value at one of the *ExSP* points. For this purpose, a set of Λ such as $\Lambda = \{0.0,\ 0.1,\ 0.2,\ \ldots,\ 1.0\}$ is defined priorily. Another way of constructing the set Λ is via line segments in $\tilde{Y}_{tN}$. Let $[y^a, y^{a+1}] \in \tilde{Y}_{tN}$ be a line segment where $y^a, y^{a+1} \in \tilde{Y}_{tXN}$ are adjacent such that $y_1^a < y_1^{a+1}$ and $y_2^a > y_2^{a+1}$. Then, $\lambda^a \in \Lambda$ is computed for all $[y^a, y^{a+1}] \in \tilde{Y}_{tN}$ as follows:

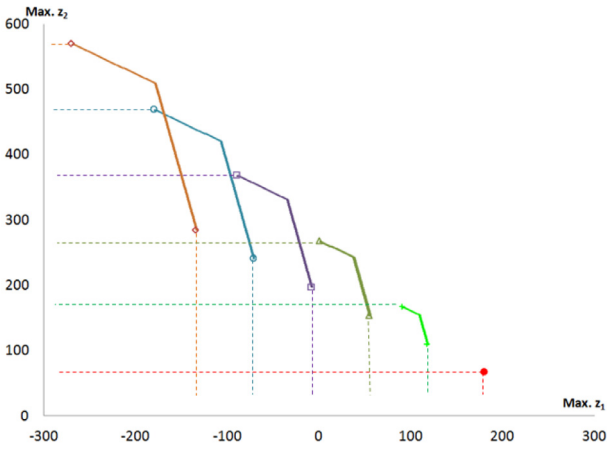
$$\lambda^a = \frac{y_2^a - y_2^{a+1}}{y_2^a - y_2^{a+1} + y_1^{a+1} - y_1^a} \quad \forall \left[y^a, y^{a+1}\right] \in \tilde{Y}_{tN} \tag{13}$$

An alternative and better way of checking condition 3 is possible with the help of another fathoming rule by Ehrgott and Gandibleux (2007) such that if for each $\breve{y} \in \breve{\theta}$, there exists a $\lambda \in \Lambda$, which satisfies $\{\lambda\breve{y}_1 + (1-\lambda)\breve{y}_2\} \geq \max_{\boldsymbol{y}\in\tilde{Y}_{tXN}}\{\lambda y_1 + (1-\lambda)y_2\}$, then condition 3 is satisfied and there is no need to further search subproblem P_t.



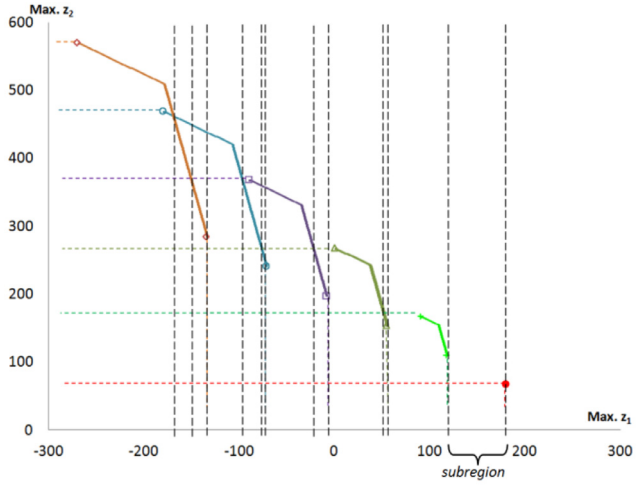
(a)

Our aim is to find the Pareto frontier of P by merging Pareto frontiers of potentially efficient slices.



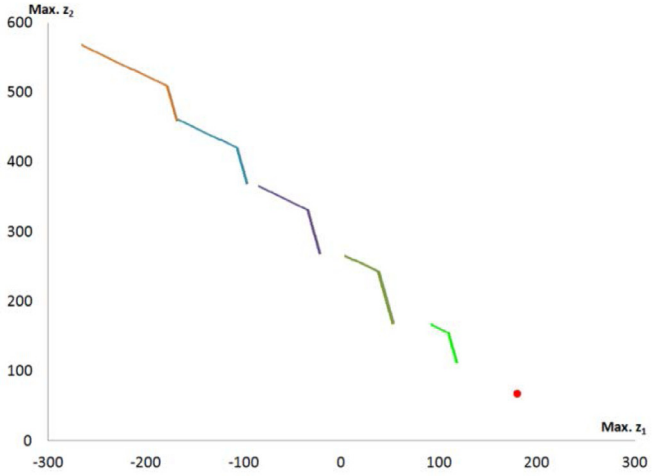
(b)

Introduce (horizontal and vertical) dominated line segments from each end point. Then, mark all intersections.



(c)

Introduce vertical lines passing through all intersection points. Subregions are constructed through adjacent lines.



(d)

Take upper line segments of each subregion. Then, remove dominated line segments. Outcome is the Pareto frontier.

Fig. 5. Finding Pareto line segments of a set of line segments.

Example 6. In Fig. 4(b) there are three line segments of the set $\tilde{Y}_{tN}$ and hence $\Lambda = \{0.40,\ 0.55,\ 0.83\}$. Note that the number of line segments in $\tilde{Y}_{tN}$ may be huge in large size problems. Considering the lower bound set in Fig. 3(a), it is impossible to satisfy condition 3 for each $\check{y} \in \tilde{\Theta}$ by using any $\lambda \in [0, 1]$.

Condition 4 can be checked similarly to condition 3. The only difference is that the MILP subproblem is considered, and hence the set $\tilde{Y}_{tXN}$ is replaced with the set Y_{tXN} while the set $\tilde{Y}_{tN}$ is replaced with the set $conv(Y_{tXN})_N$.

3.4. Constructing the Pareto frontier

The proposed SR algorithm returns a set of potentially efficient slices. First, Pareto line segments of all potentially efficient slices are computed with the parametric simplex algorithm. The Pareto frontiers of these slices might intersect and some of their points (or line segments) might be dominated (see Fig. 5(a)). In order to eliminate dominated points (or line segments) of these slices and finally returning the set of Pareto points, Soylu (2015) presented an upper envelope finding algorithm. After merging Pareto frontiers of these slices into the Pareto frontier of the original problem *P*, the SR algorithm terminates.

The main idea is to divide the objective space into consecutive, disjoint subregions and to find the upper line segments of each subregion. These subregions are unbounded over the vertical axis and their boundaries are determined in such a way that no two line segments intersect in a subregion. If these subregions are constructed through the first coordinates of intersection points, it would be guaranteed that no two line segments intersect. Another

Algorithm 1
The search-and-remove algorithm.

1: ***procedure*** *BOMILP-SR* $(P, \tilde{P})$
2: ***Set*** $\bar{X} = \emptyset$! $\bar{X}$: the set of integer valued vectors of all potentially efficient slices
$\bar{X}_t = \emptyset$! $\bar{X}_t$: the set of integer valued vectors of explored slices of P_t
$\hat{Y} = \emptyset$! $\hat{Y}$: the set of incumbent Pareto points of P
$\hat{Y}_N = \emptyset$! $\hat{Y}_N$: the set of Potentially Pareto line segments
$Y_{XN} = \emptyset$! Y_{XN}: the set of potentially *ExSP* points of P
$Y_{tXN} = \emptyset$! Y_{tXN}: the set of *ExSP* points of P_t
$\tilde{Y}_{tXN} = \emptyset$! $\tilde{Y}_{tXN}$: the set of *ExSP* points of $\tilde{P}_t$
$Y_N^* = \emptyset$! Y_N^*: the set of Pareto line segments
3: ***Set*** $\breve{\theta} = \{(-M, -M)\}$! $\breve{\theta}$: the set of local nadir points w.r.t. $\hat{Y}$
4: **Set** t=1,
Terminate = False
5: ***While*** (Terminate = =False)
6: Terminate$\leftarrow check_conditions(\tilde{P}_t, \hat{Y}, \breve{\theta}, \emptyset, \emptyset)$! check *conditions* 1 and 3
7: ***if*** Terminate == False ***then***
8: $\bar{X}, \bar{X}_t, \hat{Y}, Y_{tXN}, \breve{\theta} \leftarrow Search(P_t, \bar{X}, \emptyset, \hat{Y}, \emptyset, \breve{\theta})$! search the subproblem P_t with DS and update $\bar{X}, \hat{Y}, \breve{\theta}$
9: Terminate $\leftarrow check\_conditions(P_t, \hat{Y}, \breve{\theta}, \bar{X}_t, Y_{tXN})$! check *conditions* 2 and 4
10: ***if*** Terminate == False ***then*** ! If it could not be terminated,
11: $P_{t+1} \leftarrow P_t \cap \{H(\boldsymbol{x}, \overline{\boldsymbol{x}}) \geq 1, \quad \forall \overline{\boldsymbol{x}} \in \bar{X}_t\}$! Exclude slices by adding Tabu const.
12: $t \leftarrow t + 1$
13: ***endif***
14: ***endif***
15: ***endwhile***
16: ***for all*** $\overline{\boldsymbol{x}} \in \bar{X}$! Find line segments of all potentially efficient slices with PSA
17: $\hat{Y}_N, Y_{XN}, \emptyset, s \leftarrow PSA(P(\overline{\boldsymbol{x}}), \emptyset, \hat{Y}_N, Y_{XN})$
18: $\hat{Y}_N \leftarrow \hat{Y}_N \cup [\overleftarrow{\boldsymbol{y}}^s, \boldsymbol{y}^s] \cup [\boldsymbol{y}^1, \boldsymbol{y}^{\downarrow 1}]$! include dominated line segments for each slice
! where $\overleftarrow{\boldsymbol{y}}^s = [-M, y_2^s]$, $\boldsymbol{y}^{\downarrow 1} = [y_1^1, -M]$
19: ***endfor***
20: $Y_N^* \leftarrow ConstructPareto\ (\hat{Y}_N)$! Merge Pareto frontiers of potentially efficient slices
21: ***Return*** Y_N^*

issue is to introduce (horizontal and vertical) *dominated line segments* from each end point (Fig. 5(b)). Since each subregion will be independently evaluated, a line segment in a subregion would be aware of a point dominating itself and belonging to another subregion via *dominated line segments*. Then, all intersection points of all line segments are marked. Boundaries of subregions over the horizontal axis correspond to first coordinates of intersection points. Fig. 5(c) illustrates subregions, while Fig. 5(d) illustrates the resulting Pareto frontier.

Note that the above idea also allows computing the hypervolume measure for BOMILPs easily. The hypervolume measure was proposed by Zitzler and Thiele (1998), and it is used to compare performances of benchmark algorithms. It computes the area of the intersection of the set of dominated points (such as $Y_N^{\leqq}$) of an algorithm with the set $\breve{\breve{\boldsymbol{y}}}^{\geqq}$. In the above algorithm, if the intersected area in each subregion is computed (in Fig. 5(c)), and they are summed, then the hypervolume is obtained.

Algorithm 2
Computing ideal point.

1: ***procedure*** $ComputeIdeal\ (\tilde{P}_t)$
2: In this procedure if the subproblem P_t is introduced, then make the following replacements
Replace $\tilde{X}_t$ with X_t
3: ***If*** $\tilde{X}_t == \emptyset$ ***then*** ! the subproblem is infeasible
4: $\tilde{\boldsymbol{y}}^{tI} = (-M, -M)$
5: ***Return*** $\tilde{\boldsymbol{y}}^{tI}$
6: ***endif***
7: **Solve** $\tilde{y}_1^{tI} = max\{z_1(\tilde{\boldsymbol{x}}, \boldsymbol{x}):\ (\tilde{\boldsymbol{x}}, \boldsymbol{x}) \in \tilde{X}_t\}$
8: **Solve** $\tilde{y}_2^{tI} = max\{z_2(\tilde{\boldsymbol{x}}, \boldsymbol{x}):\ (\tilde{\boldsymbol{x}}, \boldsymbol{x}) \in \tilde{X}_t\}$
9: **Set** $\tilde{\boldsymbol{y}}^{tI} = (\tilde{y}_1^{tI}, \tilde{y}_2^{tI})$
10: ***Return*** $\tilde{\boldsymbol{y}}^{tI}$

4. The search-and-remove algorithm

In this section, the SR algorithm is presented in light of the above explanations. The SR algorithm will firstly search slices, Pareto points of which are also supported Pareto points of problem *P*. After removing them, the SR algorithm can reach slices whose Pareto points are nonsupported Pareto points of the problem *P*. The algorithm uses four conditions given in Section 3.2 to terminate. Then it is concluded that there are no more efficient slices to search. Finally, it returns a set of potentially efficient slices, and then the upper envelope algorithm computes the upper envelope of Pareto frontiers of these slices, which is the Pareto frontier of the problem.

The procedures of the proposed algorithm are given in Algorithms 1–6, where Algorithm 1 is the main one and others are subprocedures.

The inputs of Algorithm 1 are problems P and $\tilde{P}$. First, all sets are initialized to the empty set and the nadir set $\breve{\Theta}$ is initialized with a very poor point $(-M, -M)$. In line 6 of Algorithm 1, *conditions* 1 and 3 are first checked since they require solving the LP relaxed subproblem $\tilde{P}_t$. If the *check_conditions* subprocedure returns *True*, then the while loop ends. Otherwise, in line 8 the MILP subproblem P_t is searched with the DS algorithm, and the sets $\bar{X}, \hat{Y}$ and $\breve{\Theta}$ are updated with respect to new feasible solutions found. During this search, new slices associated with the *ExSP* points of subproblem P_t are explored and recorded. Now, since the *ExSP* points of subproblem P_t are available, the sets $\bar{X}_t$ and Y_{tXN} are input to the *check_conditions* subprocedure and *conditions* 2 and 4 are checked (line 9). If the termination is not possible again, all slices in $\bar{X}_t$ are excluded from the feasible space by using Tabu constraints. Actually the SR algorithm ends in line 15 and returns the set $\bar{X}$. It is guaranteed that all efficient slices are in this set. After that, line segments of all $\bar{\boldsymbol{x}} \in \bar{X}$ are constructed with the Parametric Simplex Algorithm (PSA) (lines 16–19). Then, these line segments are input to the *ConstructPareto* subprocedure, where the Pareto line segments of them are constructed with Algorithm 6 (line 20). All subprocedures are explained next.

All attainable points of the (sub)problem P_t are dominated by the ideal point, $\tilde{y}^{tI}$. Algorithm 2 (in Appendix) computes the ideal point.

Algorithm 3 checks *conditions* 1–4. Here if this procedure is called with the relaxed subproblem, then all operations are done over $\tilde{P}_t$. Otherwise, the subproblem P_t is considered and replacements given in line 2 should be done. First, the ideal point is computed in line 3. If the condition given in line 4 is satisfied, then the subproblem is infeasible and hence the algorithm immediately returns *True*. According to the condition given in line 6, if the ideal point $\tilde{\boldsymbol{y}}^{tI}$ is dominated by any attainable point available, then the procedure immediately returns *True* since *condition* 1 (*condition* 2 in case of a MILP subproblem) is satisfied. If the problem $\tilde{P}_t$ is input to this algorithm, i.e. the set $\tilde{Y}_{tN}$ is the upper bound set considered, then the subprocedure PSA is called here (in line 9) to compute the *ExSP* point of $\tilde{P}_t$. Pareto line segments of $\tilde{P}_t$ are constructed with the adjacent extreme points and stored in the set $\tilde{Y}_{tN}$. If the problem P_t is input to this algorithm, there is no need to recompute *ExSP* points of P_t (remember that *ExSP* points of P_t were computed by the *Search* subprocedure in Algorithm 1 and input to Algorithm 3). In lines 11–14, it is checked whether there is a single slice in P_t or not (see Proposition 1). If so, this slice is recorded and the procedure returns *True* since the Pareto frontier of the subproblem has already been found and nothing remains to search. In lines 15–20, *condition* 3 is evaluated for a given set of Λ. In lines 21–31, *condition* 3 is evaluated in terms of the second way.

If conditions 1 and 3 applied to the LP relaxed subproblem $\tilde{P}_t$ are not satisfied, that means it is promising to explore the MILP subproblem P_t. In Algorithm 4, the dichotomic search starts to compute *ExSP* points of P_t. Whenever a feasible point improving the parametric objective function value is found, it is introduced into the set of current Pareto points, $\hat{Y}$, and the set of *ExSP* points, Y_{tXN}. The VMAX operator is to eliminate dominated points in the set $\hat{Y}$. Two new weights λ' and λ'' are added to the set Λ. If the solution also belongs to a new slice, its integer valued vector $\boldsymbol{x}$ is added to sets $\bar{X}_t$ and $\bar{X}$. Finally, the set of local nadir points $\breve{\Theta}$ is updated with respect to $\hat{Y}$. The algorithm returns potentially efficient slices

Algorithm 5 (in Appendix) computes *ExSP* points of each subproblem $\tilde{P}_t$ by using the parametric simplex algorithm. Whenever two adjacent extreme points ($\boldsymbol{y}^{s-1}$ and $\boldsymbol{y}^s$) of $\tilde{P}_t$ are found, the Pareto line segment $[\boldsymbol{y}^{s-1}, \boldsymbol{y}^s]$ connecting them is added to the set $\tilde{Y}_{tN}$. Finally, it returns the set of integer valued vectors of explored

Algorithm 3
Checking terminate conditions.

procedure $check\_conditions(\tilde{P}_t, \tilde{Y}, \breve{\theta}, \bar{X}_t, Y_{tXN})$

In this procedure if the MILP subproblem P_t is introduced, then make the following replacements

replace $\tilde{P}_t$ with P_t, $\tilde{\boldsymbol{y}}^{tI}$ with $\boldsymbol{y}^{tI}$, $\tilde{Y}_{tXN}$ with Y_{tXN}, $\tilde{Y}_{tN}$ with $conv(Y_{tXN})_N$

$\tilde{\boldsymbol{y}}^{tI} \leftarrow ComputeIdeal(\tilde{P}_t)$

If $\tilde{\boldsymbol{y}}^{tI} == (-M, -M)$ **, then Return** *True* ! Subproblem is infeasible

endif

If $\tilde{\boldsymbol{y}}^{tI}$ is dominated by any $\boldsymbol{y} \in \hat{Y}$, **then Return** *True* ! Terminate due to the *condition* 1 (or 2)

endif

if *the relaxed subproblem* $\tilde{P}_t$ *is introduced*, **then**

$\tilde{Y}_{tN}, \tilde{Y}_{tXN}, \bar{X}_t, s \leftarrow PSA\ (\tilde{P}_t, \emptyset, \emptyset, \emptyset)$! Find ExSP points and slices of $\tilde{P}_t$ with PSA

endif

If $|\bar{X}_t| == 1$, **then** ! There is a single slice in $\tilde{P}_t$ (or P_t)

$\bar{X} \leftarrow \bar{X} \cup \bar{X}_t$

Return *True* ! There is no need to search further

endif

Set $\Lambda = \{0.0, 0.1, 0.2, \dots, 1.0\}$! first way of checking *condition* 3 (or 4)

for all $\lambda \in \Lambda$ **do**

if $max_{\boldsymbol{y} \in \tilde{Y}_{tXN}}\{\lambda y_1 + (1-\lambda) y_2\} \leq min_{\breve{y} \in \breve{\theta}}\{\lambda \breve{y}_1 + (1-\lambda)\breve{y}_2\}$, **then**

Return *True* ! There is no need to search further

endif

endfor

Set $\Lambda = \left\{\lambda^a:\ \lambda^a = \frac{y_2^a - y_2^{a+1}}{y_2^a - y_2^{a+1} + y_1^{a+1} - y_1^a} \quad \forall [\boldsymbol{y}^a, \boldsymbol{y}^{a+1}] \in \tilde{Y}_{tN}\right\}$! second way of checking condition 3 (or 4)

separated_nadir=0;

for all $\breve{\boldsymbol{y}} \in \breve{\theta}$ **do**

for all $\lambda^a \in \Lambda$ **do**

if $max_{\boldsymbol{y} \in \tilde{Y}_{tXN}}\{\lambda^a y_1 + (1-\lambda^a) y_2\} \leq \{\lambda^a \breve{y}_1 + (1-\lambda^a)\breve{y}_2\}$, **then**

separated_nadir = separated_nadir +1;

breakfor;

endif

endfor

endfor

if separated_nadir = $|\breve{\theta}|$, **then Return** *True* ! if all nadir points do not intersect with $UB^{\leqq}$, condition 3 is satisfied

Return *False*

Algorithm 4
Exploration of subproblem with dichotomic search.

1: **procedure** $Search(P_t, \bar{X}, \bar{X}_t, \hat{Y}, Y_{tXN}, \breve{\theta},)$
2: **Solve** $(\tilde{\boldsymbol{x}}^b, \boldsymbol{x}^b) \leftarrow lexmax\{z_1(\tilde{\boldsymbol{x}}, \boldsymbol{x}), z_2(\tilde{\boldsymbol{x}}, \boldsymbol{x}) : (\tilde{\boldsymbol{x}}, \boldsymbol{x}) \in X_t\}$! Solve lexicographic optimization problem
3: **Solve** $(\tilde{\boldsymbol{x}}^a, \boldsymbol{x}^a) \leftarrow lexmax\{z_2(\tilde{\boldsymbol{x}}, \boldsymbol{x}), z_1(\tilde{\boldsymbol{x}}, \boldsymbol{x}) : (\tilde{\boldsymbol{x}}, \boldsymbol{x}) \in X_t\}$
4: $Y_{tXN} \leftarrow Y_{tXN} \cup \boldsymbol{y}^a \cup \boldsymbol{y}^b$! where $\boldsymbol{y}^a = z(\tilde{\boldsymbol{x}}^a, \boldsymbol{x}^a)$ and $\boldsymbol{y}^b = z(\tilde{\boldsymbol{x}}^b, \boldsymbol{x}^b)$
5: $\hat{Y} \leftarrow \hat{Y} \cup \boldsymbol{y}^a \cup \boldsymbol{y}^b$
6: $\Lambda \leftarrow \Lambda \cup \{\frac{y_2^a - y_2^b}{y_2^a - y_2^b + y_1^b - y_1^a}\}$! Set a lambda value initially
7: $\bar{X}_t \leftarrow \bar{X}_t \cup \boldsymbol{x}^a \cup \boldsymbol{x}^b$
8: $\bar{X} \leftarrow \bar{X} \cup \boldsymbol{x}^a \cup \boldsymbol{x}^b$
9: ***For each*** $\lambda \in \Lambda$
10: ***Solve*** $(\tilde{\boldsymbol{x}}, \boldsymbol{x}) \leftarrow P_{\lambda,t}$! Solve weighted sum MILP subproblem
11: ***If*** $(\tilde{\boldsymbol{x}}, \boldsymbol{x})$ is feasible and $\lambda z_1(\tilde{\boldsymbol{x}}, \boldsymbol{x}) + (1-\lambda) z_2(\tilde{\boldsymbol{x}}, \boldsymbol{x}) > \max_{y \in Y_{tXN}} \{\lambda y_1 + (1-\lambda) y_2\}$, ***then***
12: $Y_{tXN} \leftarrow Y_{tXN} \cup \{z(\tilde{\boldsymbol{x}}, \boldsymbol{x})\}$
13: $\hat{Y} \leftarrow \hat{Y} \cup \{z(\tilde{\boldsymbol{x}}, \boldsymbol{x})\}$ and $VMAX(\hat{Y})$
14: $\Lambda \leftarrow \Lambda \cup \{\lambda', \lambda''\}$! λ' and λ'' are two new weights due to two adjacent points of $z(\tilde{\boldsymbol{x}}, \boldsymbol{x})$
15: ***If*** $\boldsymbol{x}$ belongs to a new slice, ***then***
16: $\bar{X}_t \leftarrow \bar{X}_t \cup \boldsymbol{x}$
17: $\bar{X} \leftarrow \bar{X} \cup \boldsymbol{x}$
18: ***endif***
19: ***endif***
20: ***endfor***
21: ***Update*** $\breve{\theta}$ w.r.t. $\hat{Y}$
22: ***Return*** $\bar{X}, \bar{X}_t, \hat{Y}, Y_{tXN}, \breve{\theta}$

slices, line segments and *ExSP* points of $\tilde{P}_t$. The PSA procedure is called at two points: one is to compute an upper bound (in line 9 of Algorithm 3), second is to find the line segments of all potentially efficient slices (in line 17 of Algorithm 1).

Algorithm 6 (in Appendix) is used for computing the upper envelope of potentially Pareto line segments. The input of the algorithm is the set of potentially Pareto line segments $\hat{Y}_N$ (obtained with PSA in lines 16–19 of Algorithm 1), while the output is the set of Pareto line segments Y_N^*, i.e. the Pareto frontier of P. In lines 3–10, intersection points are computed and added to a set called the IP. In lines 12–15, subregions are constructed through the IP set, and the upper envelope of each subregion is computed. Finally, the algorithm returns the set of Pareto line segments.

4.1. An illustrative example

Iterations of the SR algorithm together with bound sets are illustrated in Fig. A.1 for the example problem. Each line of Fig. A.1 represents an iteration of the SR algorithm and the algorithm terminates at the end of the fourth iteration. It can be observed from this figure that the upper bound set over the LP relaxed subproblems (i.e., i and iii in Section 3.3) are very weak and do not change after the second iteration. The lower bound set improves during iterations (except the last iteration) as new Pareto points are found (see Fig. A.1(b),(e) and (h)). The upper bound set over the MILP subproblems (i.e., ii and iv in Section 3.3) improve through iterations and finally condition 4 is satisfied at the end of the fourth iteration. At the end, the algorithm returns integer valued vectors associated with twelve slices (see the set $\bar{X}$ in Fig. A.1(k)). Six of them are efficient slices. Pareto frontiers of these twelve slices are merged to the Pareto frontier of problem *P* by using the upper envelope finding algorithm explained in Section 3.4.

4.2. Parallel processing

An extension in terms of implementation is to divide the objective space into several consecutive, disjoint subregions and to search each subregion separately. Assume that these subregions are

Algorithm 5
The parametric simplex algorithm.

procedure $PSA\ (\tilde{P}_t, \bar{X}_t, \tilde{Y}_{tN}, \tilde{Y}_{tXN})$

In this procedure if the slice problem $P(\bar{\boldsymbol{x}})$ is introduced, then make the following replacements

Replace $\tilde{P}_t$ with $P(\bar{\boldsymbol{x}})$, $\tilde{Y}_{tN}$ with $\hat{Y}_N$, $\tilde{Y}_{tXN}$ with Y_{XN}

Set $s = 1$

Solve $\tilde{P}_{\lambda,t}$ for $\lambda^s = 1$, i.e., $(\tilde{\boldsymbol{x}}^s, \boldsymbol{x}^s) \leftarrow \tilde{P}_{\lambda,t}$! Solve weighted-sum LP

$\tilde{Y}_{tXN} \leftarrow \tilde{Y}_{tXN} \cup \boldsymbol{y}^s$! $\boldsymbol{y}^s = z(\tilde{\boldsymbol{x}}^s, \boldsymbol{x}^s)$

If $(\tilde{\boldsymbol{x}}^s, \boldsymbol{x}^s)$ is feasible for P and belongs to a different slice problem, ***then***

$\bar{X}_t \leftarrow \bar{X}_t \cup \boldsymbol{x}^s$

endif

While $\lambda^s > 0$ and $\tilde{P}_{\lambda,t}$ is not unbounded

$s \leftarrow s + 1$

Compute λ^s

Solve $\tilde{P}_{\lambda,t}$ for λ^s, i.e., $(\tilde{\boldsymbol{x}}^s, \boldsymbol{x}^s) \leftarrow \tilde{P}_{\lambda,t}$! Solve weighted-sum LP

$\tilde{Y}_{tN} \leftarrow \tilde{Y}_{tN} \cup [\boldsymbol{y}^{s-1}, \boldsymbol{y}^s]$

$\tilde{Y}_{tXN} \leftarrow \tilde{Y}_{tXN} \cup \boldsymbol{y}^s$

If $(\tilde{\boldsymbol{x}}^s, \boldsymbol{x}^s)$ is feasible for P and belongs to a different slice problem, ***then***

$\bar{X}_t \leftarrow \bar{X}_t \cup \boldsymbol{x}^s$

endif

endwhile

Return $\tilde{Y}_{tN}, \tilde{Y}_{tXN}, \bar{X}_t, s$

unbounded over the horizontal axis. It is clear from the above example that for a subproblem the condition $\mathrm{UB}^{\leqq} \cap \theta^{>} = \emptyset$ is satisfied at some subregions while it is not satisfied at other regions. For instance, let us consider the subregion constructed over the interval $0 < z_2(\tilde{\boldsymbol{x}}, \boldsymbol{x}) < 100$ (see Fig. A.1(b)). Within this subregion there is only one Pareto point and this point is found at the first iteration. If this subregion was searched alone, it would be proved at the second iteration that there is no more Pareto point and hence this subregion would be eliminated. However, in the current situation the search was inessentially applied to this subregion until the end of the SR algorithm and hence a few dominated points were unnecessarily found (see Fig. A.1(e), (h), and (k)).

In order to reduce the computational effort we suggest setting small subregions at the beginning of the algorithm. These subregions can be equally sized and the SR algorithm can be applied to each subregion separately. This structure also allows parallel processing. However, if disconnected parts of the Pareto frontier can be estimated, then we could better set intervals, not have to be equally spaced.

Another extension is to update the lower bound globally although we perform search locally through each subregion. This is because of the fact that the Pareto frontier of an efficient slice might pass through a few adjacent subregions. Therefore, whenever a slice is found while searching a subregion, the lower bound is updated not only for this subregion but for whole subregions including any *ExSP* point of this slice.

5. Computational results

The computational results of the proposed SR algorithm and comparisons with benchmark algorithms from the literature are discussed in this section. The proposed algorithms were coded in C++ using the CPLEX 12.1 (CPLEX, 2010) and run on a 2.40 gigahertz workstation with 4 gigabytes of RAM memory. The performance of the SR algorithm was tested using BOMILP test problems generated by Boland et al. (2015).

In all test problems, the number of constraints and the number of variables are equal, i.e. $(m = n)$. The size of problems gradually increases up to 320 constraints and 320 variables. In all test problems, half of the n variables are set as binary, while the others are set as continuous. There are five instances of each problem size (in total 25 instances).

The results of the SR algorithm were compared with results of the triangle splitting algorithm (TSA) developed by Boland et al. (2015) and the ε, *Tabu-constraint* algorithm developed by Soylu and Yıldız (2016). The following abbreviations are used to denote performance measures.

Algorithm 6
The upper envelope finding algorithm.

```
1: procedure ConstructPareto(Ŷ_N)
2:     Set IP = ∅,                                   ! IP : the set of intersection points
3:     For all [y^j, y^{j+1}] ∈ Ŷ_N  do             ! Take a line segment [y^j, y^{j+1}] ∈ Ŷ_N
4:         For all [y^r, y^{r+1}] ∈ Ŷ_N , r > j  do  ! Compare it with others
5:             if [y^j, y^{j+1}] and [y^r, y^{r+1}] intersects, then   ! within the [y_1^j − y_1^{j+1}] range
6:                 Compute intersection point y^i
7:                 IP ← IP ∪ y^i
8:             endif
9:         endfor
10:    endfor
11:    Sort y^i ∈ IP in nondecreasing order of the first coordinate
12:    For i=0 : |IP|  do
13:        Find the upper line segment [y^i, y^{i+1}]* in the [y_1^i, y_1^{i+1}] subregion
14:        Y_N^* ← Y_N^* ∪ [y^i, y^{i+1}]*
15:    endfor
16: Return Y_N^*
```

CPU: the CPU time (measured in seconds) of the SR algorithm.
CPU-A6: the CPU time (measured in seconds) of Algorithm 6, i.e. the upper envelope finding algorithm.
CPU-mip: the CPU time (measured in seconds) to solve MILP models.
CPU-LP: the CPU time (measured in seconds) to solve LP models.
#milp: number of MILP models solved.
#LP: number of LP models solved.
#Tabu: number of Tabu constraints added.
#it: number of iterations performed.
#subregion: number of subregions processed.
%ES: the percentage of efficient slices found at each iteration.
SingS∼: #of times the SR algorithm terminates due to single slice in $\tilde{P}_t$.
SingS: #of times the SR algorithm terminates due to single slice in P_t.
Cnd1: #of times the SR algorithm terminates due to condition 1.
Cnd2: #of times the SR algorithm terminates due to condition 2.
Cnd3: #of times the SR algorithm terminates due to condition 3.
Cnd4: #of times the SR algorithm terminates due to condition 4.

Table 1 presents the CPU time results of the SR algorithm on the test problems by Boland et al. (2015) for different number of subregions. Here in case of *#subregion* = 1 the objective space was not divided into any subregion while in case of *#subregion* = 10 the objective space was divided into ten equally spaced consecutive, disjoint subregions over the range of the first objective function. Each subregion is processed sequentially and the total CPU time is presented. It is especially observed for C80, 160 and 320 instances that as the number of subregions increases the CPU time decreases in general. Accordingly, the best performing *#subregion* setting is 10 among all. As is expected the highest portion of the CPU time of the algorithm is due to the MILP solution time (CPU-milp).

The performances of the SR algorithm, the TSA (Boland et al., 2015) and the ε, *Tabu-constraint* algorithm (Soylu & Yıldız, 2016) are compared in Table 2. The best performing algorithm in terms of CPU and *#milp* is marked as bold. It can be observed that the SR algorithm is more advantageous in mid and large size instances. When we analyze C320 instances, we observe that the SR algorithm performs better in terms of CPU time (except inst.23) while the TSA performs better in terms of *#milp* measure. Thanks to subregions constructed, the SR algorithm solves higher number of MILPs with less CPU time than the TSA algorithm. Searching smaller subspaces requires less computational effort. If different parallel processors worked for each subregion, we would expect much promising performance from the SR algorithm.

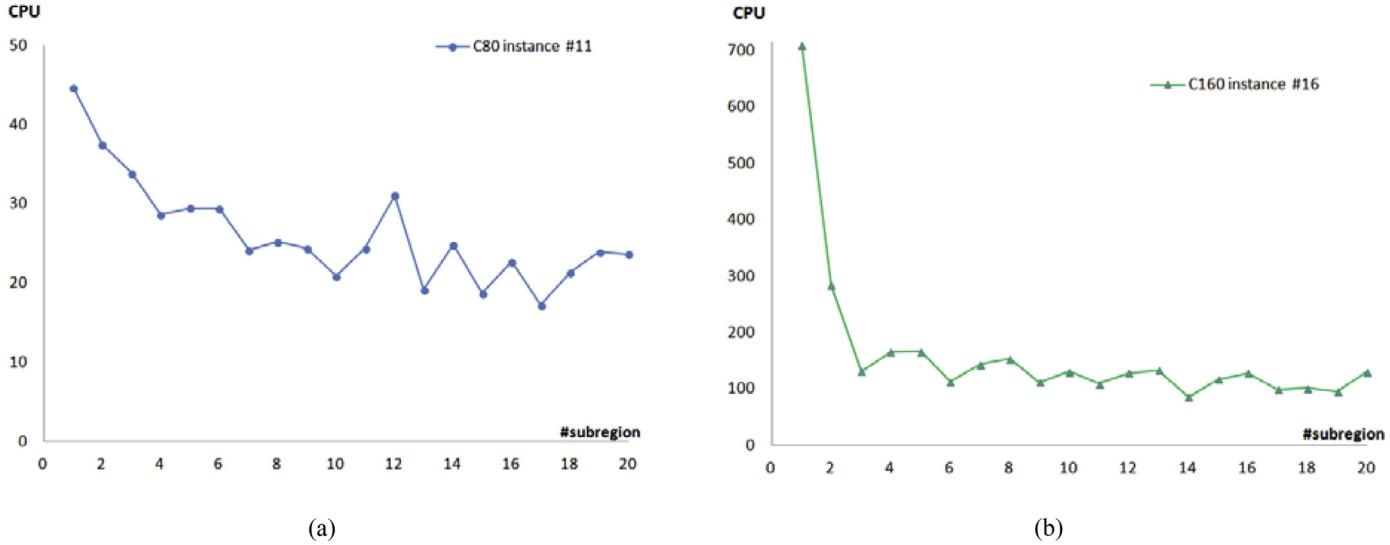
Fig. 6 illustrates the number of subregions versus the CPU time results for two different size instances. Here the number of subregions was set from 1 to 20, and the CPU time of the SR algorithm was recorded for each setting. In Fig. 6(a), we observe a decrement in the CPU time until *#subregion* = 10. In Fig. 6(b), a sharp decrement is observed until *#subregion* = 3. After that point small ups and downs appear in the CPU time. The trend in other problems is also analogous. This figure signifies first that the decremental trend in the CPU time through the number of subregions is limited. Secondly, selecting the number of subregions around 10 (consistent with Table 1 results as well) might reduce the computational effort of the SR algorithm.

Table 1
The CPU time results of the SR algorithm for different number of subregions.

		#subregion = 1			*#subregion* = 2			*#subregion* = 3			*#subregion* = 4			*#subregion* = 5			*#subregion* = 10		
Problem		CPU	CPU-mip	CPU-LP	CPU	CPU-mip	CPU-LP	CPU	CPU-mip	CPU-LP	CPU	CPU-mip	CPU-LP	CPU	CPU-mip	CPU-LP	CPU	CPU-mip	CPU-LP
C20	inst.1	0.6	0.5	0.0	0.2	0.2	0.0	0.8	0.8	0.0	1.2	1.1	0.0	0.7	0.6	0.0	1.0	0.7	0.0
	inst.2	3.2	3.2	0.0	5.5	5.4	0.0	3.9	3.8	0.1	4.9	4.7	0.0	5.6	5.5	0.1	5.3	5.1	0.0
$n = 20$	inst.3	2.6	2.5	0.0	2.4	2.4	0.0	4.5	4.4	0.0	3.2	3.2	0.0	3.9	3.9	0.0	3.0	2.9	0.0
$p = 10$	inst.4	4.7	4.6	0.0	5.0	4.9	0.0	3.1	3.0	0.0	4.6	4.5	0.0	6.7	6.6	0.1	5.3	5.1	0.0
	inst.5	1.2	1.1	0.0	1.6	1.5	0.0	3.0	3.0	0.0	2.4	2.3	0.0	1.7	1.7	0.0	2.2	2.1	0.0
	Avg.	2.5	2.4	0.0	2.9	2.9	0.0	3.1	3.0	0.0	3.3	3.2	0.0	3.7	3.7	0.0	3.4	3.2	0.0
	Max.	4.7	4.6	0.0	5.5	5.4	0.0	4.5	4.4	0.1	4.9	4.7	0.0	6.7	6.6	0.1	5.3	5.1	0.0
C40	inst.6	4.7	4.2	0.3	6.6	6.1	0.4	4.9	4.2	0.4	6.1	5.3	0.5	5.4	4.4	0.7	4.6	3.5	0.6
	inst.7	0.9	0.7	0.1	0.9	0.7	0.1	1.0	0.7	0.1	1.3	1.1	0.1	1.3	1	0.2	1.5	1.1	0.2
$n = 40$	inst.8	3.6	3.4	0.2	2.0	1.8	0.2	2.4	2.2	0.2	2.2	1.9	0.2	3.0	2.5	0.3	2.6	2	0.2
$p = 20$	inst.9	2.7	2.4	0.3	2.3	2.0	0.3	2.7	2.3	0.3	2.7	2.2	0.4	2.1	1.6	0.2	2.5	1.8	0.4
	inst.10	2.7	2.4	0.3	2.3	2.0	0.2	1.1	0.9	0.1	1.2	0.9	0.2	1.4	1	0.2	1.4	0.9	0.3
	Avg.	2.9	2.6	0.2	2.8	2.5	0.2	2.4	2.1	0.2	2.7	2.3	0.3	2.6	2.1	0.3	2.5	1.9	0.3
	Max.	4.7	4.2	0.3	6.6	6.1	0.4	4.9	4.2	0.4	6.1	5.3	0.5	5.4	4.4	0.7	4.6	3.5	0.6
C80	inst.11	44.7	40.0	4.7	37.5	31.8	4.8	33.9	28.3	4.4	28.6	23.4	4.2	29.5	25.1	3.5	20.9	16.8	3.3
	inst.12	40.0	35.3	4.0	39.0	34.5	4.9	34.1	29.7	4.6	30.4	25.6	4.3	23.5	19.5	3.3	18.1	13.9	3.4
$n = 80$	inst.13	127.3	119.3	5.8	70.3	59.4	7.7	35.7	29.3	5.0	39.4	32.4	5.2	40.4	33.2	5.2	29.9	23.4	5.0
$p = 40$	inst.14	39.9	35.9	4.0	26.7	23.2	3.4	23.6	20.3	3.4	31.7	26.6	4.8	27.5	23.0	3.9	26.6	21.4	4.2
	inst.15	30.6	26.5	3.9	27.1	23.5	4.1	27.6	23.4	4.1	21.2	18.0	3.1	28.3	23.5	4.1	19.5	15.5	3.3
	Avg.	56.5	51.4	4.5	40.1	34.5	5.0	31.0	26.2	4.3	30.3	25.2	4.3	29.8	24.9	4.0	23.0	18.2	3.8
	Max.	127.3	119.3	5.8	70.3	59.4	7.7	35.7	29.7	5.0	39.4	32.4	5.2	40.4	33.2	5.2	29.9	23.4	5.0
C160	inst.16	710.2	630.1	60.0	285.0	239.4	34.1	132.1	111.1	17.9	165.7	139.1	23.2	166.5	135.5	25.3	131.2	99.2	29.5
	inst.17	2060.8	1916.0	110.1	1082.3	993.4	77.1	510.7	447.9	55.1	297.8	254.1	38.6	127.6	97.3	30.0	152.6	115.1	35.5
$n = 160$	inst.18	225.4	191.0	23.7	149.0	125.0	18.7	158.1	133.2	21.1	151.8	124.8	22.0	208.6	167.1	32.8	152.4	122.1	25.1
$p = 80$	inst.19	1055.5	923.1	81.0	1136.5	994.7	108.8	503.3	432.5	58.3	562.4	485.6	64.6	484.5	408.5	63.5	407.3	331.8	61.2
	inst.20	252.7	226.3	19.0	157.7	139.7	14.8	202.8	179.1	19.6	140.1	122.1	15.3	142.0	124.5	14.5	125.9	104.4	20.2
	Avg.	860.9	777.3	58.8	562.1	498.4	50.7	301.4	260.8	34.4	263.6	225.1	32.7	225.8	186.6	33.2	193.9	154.5	34.3
	Max.	2060.8	1916.0	110.1	1136.5	994.7	108.8	510.7	447.9	58.3	562.4	485.6	64.6	484.5	408.5	63.5	407.3	331.8	61.2
C320	inst.21	9910.9	8249.3	1007.7	5065.5	4092.7	761.8	4942.2	4030.4	773.7	4557.2	3690.2	761.6	4969.5	4074.1	801.2	2676.7	2149.8	484.5
	inst.22	28336.9	24941.3	1930.0	15863.8	13744.7	1591.2	6610.6	5534.9	827.2	9899.8	8544.3	1141.8	5559.7	4605.2	803.3	3017.9	2430.8	535.9
$n = 320$	inst.23	32705.9	29156.9	2074.0	19528.8	17445.9	1547.1	12209.2	10709.8	1219.1	11689.3	10280.3	1185.9	10244.8	8965.2	1100.3	7973.4	6985.3	898.0
$p = 160$	inst.24	61592.8	56108.6	3000.8	37060.8	33869.2	2264.6	25247.7	22878.8	1828	20182	18285.7	1532.4	7029.7	5968	894.2	3905.7	3183.7	659.2
	inst.25	8481.6	7575.6	799.9	5245.9	4484.9	698.4	3867.4	3195.4	626.8	4455.8	3596.1	812.3	2870.2	2369.2	469.2	552.1	397.8	143.3
	Avg.	28205.6	25206.3	1762.5	16553.0	14727.5	1372.6	10575.4	9269.9	1055.0	10156.8	8879.3	1086.8	6134.8	5196.3	813.6	3625.2	3029.5	544.2
	Max.	61592.8	56108.6	3000.8	37060.8	33869.2	2264.6	25247.7	22878.8	1828.0	20182.0	18285.7	1532.4	10244.8	8965.2	1100.3	7973.4	6985.3	898.0

Table 2
Comparing performances of TSA, ε, *Tabu*-constraint and SR algorithms.

		TSA		ε, Tabu-const.		SR (*#subregion* = 10)	
Problem		CPU	#milp	CPU	#milp	CPU	#milp
C20	inst.1	0.3	70	**0.2**	**33**	1.0	57
	inst.2	0.7	171	**0.4**	**76**	5.3	150
$n = 20$	inst.3	0.7	151	**0.5**	**66**	3.0	81
$p = 10$	inst.4	1.0	200	**0.5**	**72**	5.3	164
	inst.5	0.3	74	**0.2**	**37**	2.2	69
	Avg.	0.6	133.2	**0.4**	**56.8**	3.4	104.2
	Max.	1.0	200	**0.5**	**76**	5.3	164
C40	inst.6	5.4	763	**3.2**	**380**	4.6	550
	inst.7	1.9	318	**1.0**	**124**	1.5	190
$n = 40$	inst.8	2.3	351	**1.2**	**153**	2.6	246
$p = 20$	inst.9	2.2	392	**1.4**	**207**	2.5	318
	inst.10	1.9	319	**1.2**	**142**	1.4	188
	Avg.	2.8	428.6	**1.6**	**201.2**	2.5	298.4
	Max.	5.4	763	**3.2**	**380**	4.6	550
C80	inst.11	31.6	1700	21.8	1118	**20.9**	**1018**
	inst.12	21.1	1245	**12.6**	**752**	18.1	964
$n = 80$	inst.13	36.1	1920	**21.3**	**1024**	29.9	1034
$p = 40$	inst.14	34.4	1841	**18.6**	1008	26.6	**1004**
	inst.15	22.1	1342	**11.7**	**801**	19.5	823
	Avg.	29.1	1609.6	**17.2**	**940.6**	23.0	968.6
	Max.	36.1	1920	**21.8**	1118	29.9	**1034**
C160	inst.16	203.5	2470	152.5	2888	**131.2**	**2367**
	inst.17	220.1	**2334**	203.4	3070	**152.6**	2425
$n = 160$	inst.18	209.6	**2260**	169.3	2844	**152.4**	2757
$p = 80$	inst.19	507.9	**4593**	494.6	6353	**407.3**	5583
	inst.20	231.6	2541	214.8	3226	**125.9**	**2224**
	Avg.	274.5	**2839.6**	246.9	3676.2	**193.9**	3071.2
	Max.	507.9	**4593**	494.6	6353	**407.3**	5583
C320	inst.21	3121.0	**4490**	7171.8	19003	**2676.7**	10835
	inst.22	4569.2	**6403**	7413.9	20049	**3017.9**	9539
$n = 320$	inst.23	**3741.8**	**5167**	5650.4	13128	7973.4	15555
$p = 160$	inst.24	4661.2	**6016**	5954.2	7445	**3905.7**	12005
	inst.25	3170.0	4865	4641.4	13908	**552.1**	**2827**
	Avg.	3852.6	**5388.2**	6166.3	14707	**3625.2**	10152.2
	Max.	**4661.2**	**6403**	7413.9	20049	7973.4	15555



(a) (b)

Fig. 6. The number of subregions versus the CPU time results.

Table 3 presents several statistics about which condition stops the SR algorithm at each subregion. Although these statistics are given for *#subregion* = 3, 4, 5 and 10 due to limited column space, the trend is also similar for *#subregion* = 1 and 2. According to these results, the most effective stopping condition is condition 4. Especially for mid and large size problems, nearly in all subregions the SR algorithm stops due to condition 4 indicating the effectiveness of this condition. In small size problems, condition 2 or 3 also stops the SR algorithm. Here another issue is the existence of the Pareto frontier of a single slice in a subregion where we have already got the Pareto frontier of the subproblem (due to Proposition 1). Especially in C20 problems this case is detected by

Table 3
Statistics about which condition stops the SR algorithm.

		#subregion = 3						#subregion = 4						#subregion = 5						#subregion = 10					
Problem	SingS~	SingS	Cnd1	Cnd2	Cnd3	Cnd4		SingS~	SingS	Cnd1	Cnd2	Cnd3	Cnd4	SingS~	SingS	Cnd1	Cnd2	Cnd3	Cnd4	SingS~	SingS	Cnd1	Cnd2	Cnd3	Cnd4
C20	Avg.	0	0.2	0	0.6	0.2	1.8	0	0.6	0	1.4	0.2	1.8	0.2	0.8	0	1.6	0.2	2.2	0.4	4.4	0.2	3.8	0	1.2
	Max.	0	1	0	2	1	3	0	2	0	3	1	3	1	3	0	3	1	5	2	7	1	7	0	4
C40	Avg.	0	0	0	0	0	3	0	0	0	0.6	0	3.4	0	0	0	0.6	0	4.4	0	1.6	0	1.4	0	7
	Max.	0	0	0	0	0	3	0	0	0	1	0	4	0	0	0	1	0	5	0	3	0	2	0	9
C80	Avg.	0	0	0	0	0	3	0	0	0	0	0	4	0	0	0	0	0	5	0	0	0	0.2	0	9.8
	Max.	0	0	0	0	0	3	0	0	0	0	0	4	0	0	0	0	0	5	0	0	0	1	0	10
C160	Avg.	0	0	0	0	0	3	0	0	0	0	0	4	0	0	0	0	0	5	0	0	0	0	0	10
	Max.	0	0	0	0	0	3	0	0	0	0	0	4	0	0	0	0	0	5	0	0	0	0	0	10
C320	Avg.	0	0	0	0	0	3	0	0	0	0	0	4	0	0	0	0	0	5	0	0	0	0	0	10
	Max.	0	0	0	0	0	3	0	0	0	0	0	4	0	0	0	0	0	5	0	0	0	0	0	10

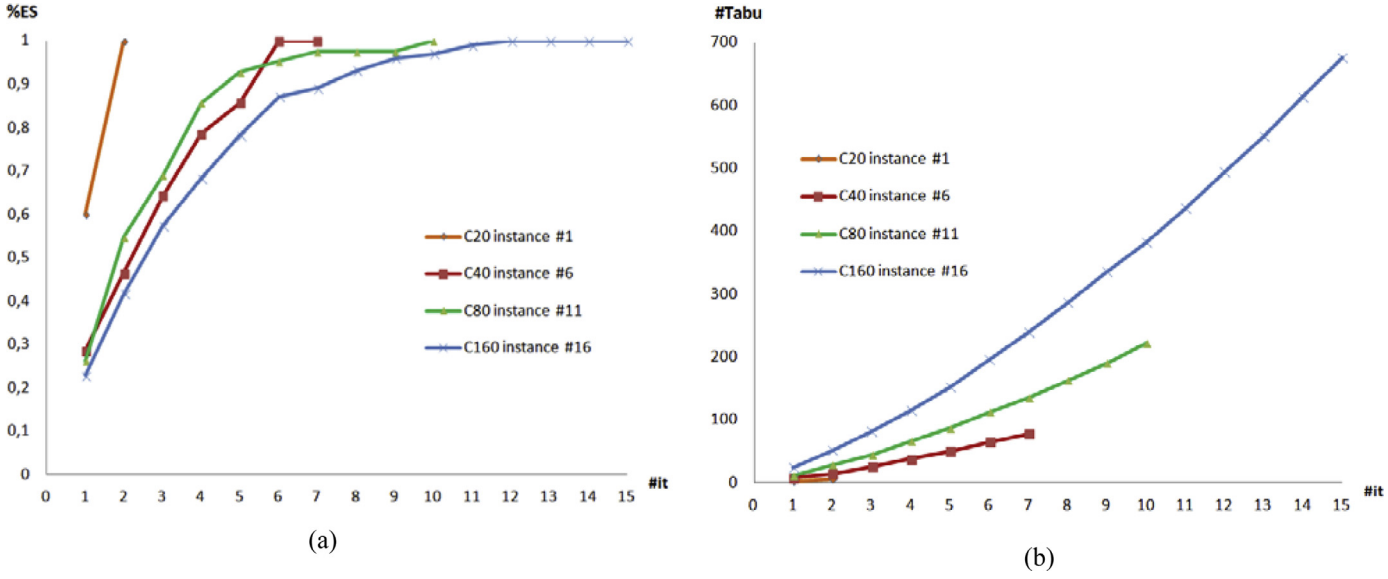


Fig. 7. Statistics for the SR algorithm (#subregion = 1) over iterations (a) the percentage of efficient slices found (b) the number of Tabu constraints added.

the algorithm in some subregions. As the number of subregions increases, the single slice detection increases. This is natural because as the subregion size decreases, the probability of observing different Pareto frontiers associated with different slices decreases in a subregion.

Fig. 7(a) illustrates the percentage of efficient slices found at each iteration of the SR algorithm for different sizes of problem instances. The %ES statistic has an increasing trend with a decreasing rate. The trend in other problems is also analogous. After finding all efficient slices, the SR algorithm performs a few more iterations to prove this. Fig. 7(b) illustrates the size of the Tabu constraint set during iterations. In C160 instance where there are 160 variables and 160 constraints, the number of Tabu constraints added is 676 at the end of the SR algorithm. That means 676 slices excluded with the help of Tabu constraints. We know that 115 out of 676 slices are efficient slices. The remaining ones are excluded due either to reach nonsupported efficient slices or to prove that all efficient slices are found.

6. Conclusion

In this study, a new exact algorithm was presented for BOMILP problems. The Pareto frontier of BOMILP might include Pareto line segments and isolated Pareto points. This property makes it difficult finding the Pareto frontier with classical approaches such as the ε-constraint algorithm, which is able to provide a subset of efficient solutions. This research area is relatively new. As different from other studies in the literature, the proposed algorithm finds the Pareto frontier by considering Pareto frontiers of (BOLP) slice problems. As the slice problems are searched, they are removed from the feasible solutions set until proving that the whole set of efficient slices is found. It is also shown that working with subregions improves the performance of the algorithm and allows parallelization. Furthermore, experimental results on test problems from the literature confirm that the SR algorithm can solve fairly large-size instances within a reasonable time.

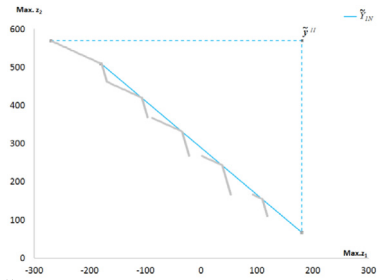
Developing a tighter lower bound set is one of the future research directions. As indicated in Fig. 3 above, keeping feasible line segments results in a tighter lower bound set and helps to reduce the iterations of the algorithm. At this point the upper envelope finding algorithm, i.e. Algorithm 6, can be used at each iteration to update the lower bound set.

Acknowledgment

This study was supported by TUBITAK (The Scientific and Technological Research Council of Turkey) under grant no 315M109.

Appendix

Iteration t=1:



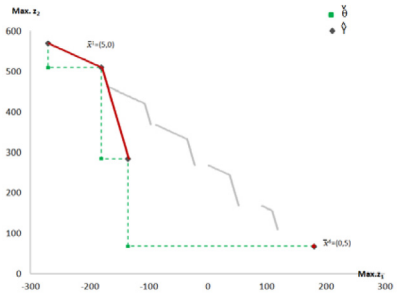
$\breve{\theta} = \{(-\infty, -\infty)\}$

$\tilde{y}^{1l} = (180, 570)$

$\tilde{Y}_{1N} = \left\{ \begin{matrix} [(180{,}67.5), (-180{,}510)], \\ [(-180{,}510), (-270{,}570)] \end{matrix} \right\}$

Since *conditions* 1 or 3 are not satisfied, continue with the *Search* procedure.

(*a*)

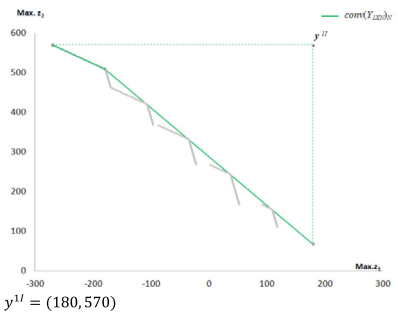


After search with the DS alg. over P_1,
$Y_{1XN} = \{(180{,}67.5), (-180, 510), (-270, 510)\}$

$\bar{X}_1 = \{(5, 0), (0, 5)\}$
$\bar{X} = \{(5, 0), (0, 5)\}$

After PSA over $\bar{X}_1$
$\hat{Y} = \{(180{,}67.5), (-135, 285), (-180, 510), (-270, 510)\}$
$\hat{\theta} = \{(-270{,}510), (-180{,}285), (-135{,}67.5)\}$

(*b*)



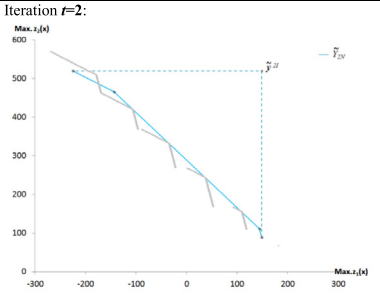
$y^{1l} = (180, 570)$

$conv(Y_{1XN})_N = \left\{ \begin{matrix} [(180{,}67.5), (-180{,}510)], \\ [(-180{,}510), (-270{,}570)] \end{matrix} \right\}$

Since *conditions* 2 or 4 are not satisfied, add Tabu constraints $H(\boldsymbol{x}, \bar{\boldsymbol{x}}) \geq 1 \quad \forall \bar{\boldsymbol{x}} \in \bar{X}_1$

(*c*)

Iteration t=2:

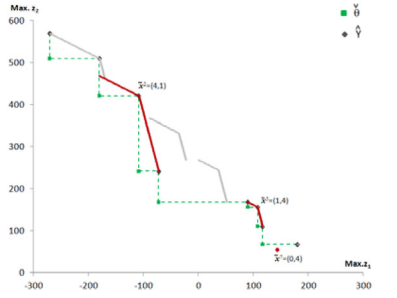


$\tilde{y}^{2l} = (148.5, 519.75)$

$\tilde{Y}_{2N} = \left\{ \begin{matrix} [(148.5, 89.25), (144, 111.75)], \\ [(-144, 465.75), (-225, 519.75)] \end{matrix} \right\}$

Since *conditions* 1 or 3 are not satisfied, continue with the *Search* procedure.

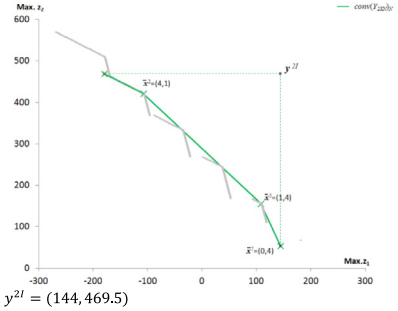
(*d*)



After search with the DS alg. over P_2,
$Y_{2XN} = \{(144, 54), (108, 56), (-108, 421.5), (-180, 469.5)\}$

$\bar{X}_2 = \{(4, 1), (1, 4), (0, 4)\}$
$\bar{X} = \{(5, 0), (0, 5), (4, 1), (1, 4), (0, 4)\}$
After PSA over $\bar{X}_2$
$\hat{Y} = \{(180{,}67.5), (117, 111), (108, 156), \ldots, (-270, 510)\}$
$\hat{\theta} = \{(-270{,}510), (-180{,}421.5), \ldots, (117{,}67.5)\}$

(*e*)



$y^{2l} = (144, 469.5)$

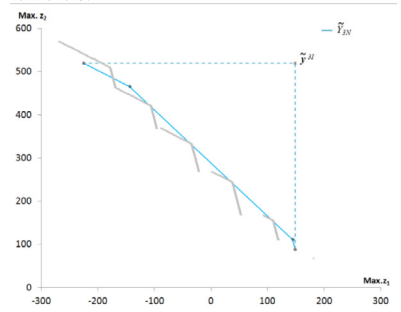
$conv(Y_{2XN})_N = \left\{ \begin{matrix} [(144{,}54), (108{,}56)], \ldots, \\ [(-108{,}421.5), (-180{,}469.5)] \end{matrix} \right\}$

Since *conditions* 2 or 4 are not satisfied, add Tabu constraints $H(\boldsymbol{x}, \bar{\boldsymbol{x}}) \geq 1 \quad \forall \bar{\boldsymbol{x}} \in \bar{X}_2$

(*f*)

Fig. A.1. Iterations of the SR algorithm over an example.

Iteration t=3:

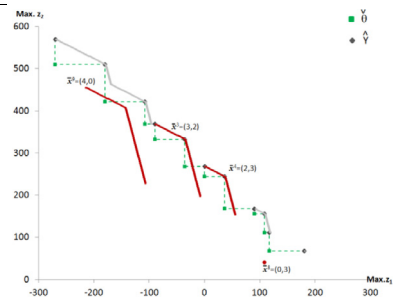


$\tilde{y}^{3I} = (148.5, 519.75)$

$\tilde{Y}_{3N} = \left\{ \begin{matrix} [(148.5, 89.25), (144, 111.75)], \\ [(-144, 465.75), (-225, 519.75)] \end{matrix} \right\}$

Since *conditions* 1 or 3 are not satisfied, continue with the *Search* procedure.

(*g*)



After search with the DS alg. P_3,

$Y_{3XN} = \left\{ \begin{matrix} (108, 40.5), (36, 244.5), (-36, 333) \\ (-144, 408), (-216, 456) \end{matrix} \right\}$

$\bar{X}_3 = \{(0,3), (2,3), (3,2), (4,0)\}$

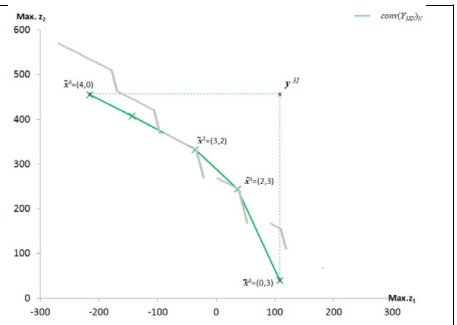
$\bar{X} = \left\{ \begin{matrix} (5,0), (0,5), (4,1), (1,4), (0,4), \\ (0,3), (2,3), (3,2), (4,0) \end{matrix} \right\}$

After PSA over $\bar{X}_3$,

$\hat{Y} = \{(180, 67.5), (117, 111), (108, 156), \ldots, (-270, 510)\}$

$\check{\theta} = \{(-270, 510), (-180, 421.5), \ldots, (117, 67.5)\}$

(*h*)



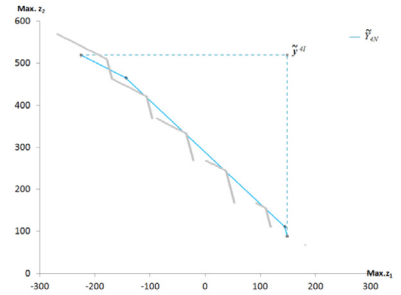
$y^{3I} = (40.5, 456)$

$conv(Y_{3XN})_N = \left\{ \begin{matrix} [(108, 40.5), (36, 244.5)], \ldots, \\ [(-144, 408), (-216, 456)] \end{matrix} \right\}$

Since *conditions* 2 or 4 are not satisfied, add Tabu constraints $H(\boldsymbol{x}, \bar{\boldsymbol{x}}) \geq 1 \quad \forall \bar{\boldsymbol{x}} \in \bar{X}_3$

(*i*)

Iteration t=4:

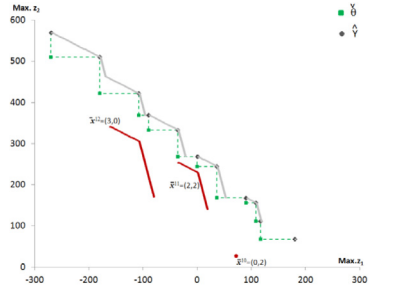


$\tilde{y}^{3I} = (148.5, 519.75)$

$\tilde{Y}_{3N} = \left\{ \begin{matrix} [(148.5, 89.25), (144, 111.75)], \\ [(-144, 465.75), (-225, 519.75)] \end{matrix} \right\}$

Since *conditions* 1 or 3 are not satisfied, continue with the *Search* procedure.

(*j*)



After search with the DS alg. over P_4,

$Y_{4XN} = \{(72, 27), (0, 231), (-108, 306), (-162, 342)\}$

$\bar{X}_4 = \{(3,0), (0,2), (2,2)\}$

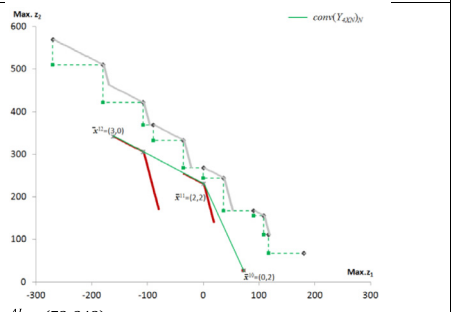
$\bar{X} = \left\{ \begin{matrix} (5,0), (0,5), (4,1), (1,4), (0,4), (0,3), \\ (2,3), (3,2), (4,0), (3,0), (0,2), (2,2) \end{matrix} \right\}$

After PSA over $\bar{X}_4$,

$\hat{Y} = \{(180, 67.5), (117, 111), (108, 156), \ldots, (-270, 510)\}$

$\check{\theta} = \{(-270, 510), (-180, 421.5), \ldots, (117, 67.5)\}$

(*k*)



$y^{4I} = (72, 342)$

$conv(Y_{4XN})_N = \left\{ \begin{matrix} [(72, 27), (0, 231)], \ldots, \\ [(-108, 306), (-162, 342)] \end{matrix} \right\}$

There exists a $\lambda \in \Lambda$, which satisfies *Condition* 4. STOP. The upper envelope of the Pareto frontiers of slices associated with $\bar{X}$, which is the Pareto frontier of P, is computed with Algorithm 6.

(*l*)

Fig. A.1. (Continued).

References

Ahmadi, A., Aghaei, J., Shayanfar, H. A., & Rabiee, A. (2012). Mixed integer programming of multiobjective hydro-thermal self-scheduling. *Applied Soft Computing, 12*(8), 2137–2146.

Aneja, Y. P., & Nair, K. P. K. (1979). Bicriteria transportation problem. *Management Science, 25*, 73–78.

Belotti, P., Soylu, B., & Wiecek, M (2013). *A branch-and-bound algorithm for biobjective mixed-integer programs* Available online at http://www.optimization-online.org/DB_FILE/2013/01/3719.pdf.

Belotti, P., Soylu, B., & Wiecek, M. M. (2016). Fathoming rules for biobjective mixed integer linear programs: Review and extensions. *Discrete Optimization, 22*, 341–363.

Bitran, G. R., & Rivera, J. M. (1982). A combined approach to solve binary multicriteria problems. *Naval Research Logistics, 29*(2), 181–201.

Boland, N., Charkhgard, H., & Savelsbergh, M. (2015). Criterion space search algorithm for biobjective mixed integer programming: The triangle splitting method. *INFORMS Journal on Computing, 27*(4), 597–618.

Cohon, J. L. (1978). *Multiobjective programming and planning*. New York: Academic Press.

CPLEX (2010). IBM ILOG CPLEX 12.1 optimizer user's manual.

Delort, C., & Spanjaard, O. (2010). Using bound sets in multiobjective optimization: Application to the biobjective binary knapsack problem. *Experimental algorithms* (pp. 253–265). Springer Berlin Heidelberg.

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

Ehrgott, M., & Gandibleux, X. (2007). Bound sets for biobjective combinatorial optimization problems. *Computers & Operations Research, 34*(9), 2674–2694.

Fischetti, M., & Lodi, A. (2003). Local branching. *Mathematical Programming Series B, 98*, 23–47.

Fischetti, M., Glover, F., & Lodi, A. (2005). The feasibility pump. *Mathematical Programming, 104*(1), 91–104.

Gadegaard, S., Ehrgott, M., & Nielsen, L. (2016). *Bi-objective branch-and-cut algorithms: Applications to the single source capacitated facility location problem* Available online at http://www.optimization-online.org/DB_FILE/2016/04/5402.pdf.

Haimes, Y. Y., Lasdon, L., & Wismer, D. (1971). On a bicriterion formulation of the problems of integrated system identification and system optimization. *IEEE Transactions on Systems, Man, and Cybernetics, 1*(3), 296–297.

Hamming, R. W. (1950). Error detecting and error correcting codes. *Bell System Technical Journal, 29*(2), 147–160.

Hooker, J. (2011). *Logic-based methods for optimization: Combining optimization and constraint satisfaction*: vol. 2. John Wiley & Sons.

Isermann, H. (1974). Proper efficiency and the linear vector maximum problem. *Operations Research, 22*, 189–191.

Köksalan, M., & Soylu, B. (2010). Bicriteria p-hub location problems and evolutionary algorithms. *INFORMS Journal on Computing, 22*(4), 528–542.

Mavrotas, G., & Diakoulaki, D. (1998). A branch and bound algorithm for mixed zero-one multiple objective linear programming. *European Journal of Operational Research, 107*(3), 530–541.

Mavrotas, G., Diakoulaki, D., & Papayannakis, L. (1999). An energy planning approach based on mixed 0–1 multiple objective linear programming. *International Transactions in Operational Research, 6*(2), 231–244.

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

Naccache, P. H. (1978). Connectedness of the set of nondominated outcomes in multicriteria optimization. *Journal of Optimization Theory and Applications, 25*(3), 459–467.

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

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

Rong, A., & Figueira, J. R. (2014). Dynamic programming algorithms for the bi-objective integer knapsack problem. *European Journal of Operational Research, 236*(1), 85–99.

Rong, A., Figueira, J. R., & Lahdelma, R. (2015). A two phase approach for the bi-objective non-convex combined heat and power production planning problem. *European Journal of Operational Research, 245*(1), 296–308.

Rudloff, B., Ulus, F., & Vanderbei, R. (2017). A parametric simplex algorithm for linear vector optimization problems. *Mathematical Programming, 163*(1–2), 213–242.

Sourd, F., & Spanjaard, O. (2008). A multiobjective branch-and-bound framework: Application to the biobjective spanning tree problem. *INFORMS Journal on Computing, 20*(3), 472–484.

Soylu, B. (2015). Heuristic approaches for biobjective mixed 0–1 integer linear programming problems. *European Journal of Operational Research, 245*(3), 690–703.

Soylu, B., & Yıldız, G. B. (2015). *A local-branch-and-bound algorithm for biobjective mixed integer linear programming problems*. Department of Industrial Engineering, Erciyes University, Technical report.

Soylu, B., & Yıldız, G. B. (2016). An exact algorithm for biobjective mixed integer linear programming problems. *Computers & Operations Research, 72*, 204–213.

Steuer, R. (1985). *Multiple criteria optimization: Theory, computation and application*. New York, NY: John Wiley & Sons.

Stidsen, T., Andersen, K. A., & Dammann, B. (2014). A branch and bound algorithm for a class of biobjective mixed integer programs. *Management Science, 60*(4), 1009–1032.

Villarreal, B., & Karwan, M. H. (1981). Multicriteria integer programming: A (hybrid) dynamic programming recursive approach. *Mathematical Programming, 21*(1), 204–223.

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. *Computers & Operations Research, 40*(1), 498–509.

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

Yu, P. L., & Zeleny, M. (1975). The set of all nondominated solutions in linear cases and a multicriteria simplex method. *Journal of Mathematical Analysis and Applications, 49*(2), 430–468.

Zitzler, E., & Thiele, L. (1998). Multiobjective optimization using evolutionary algorithms—A comparative case study. In *Proceedings of the 1998 international conference on parallel problem solving from nature* (pp. 292–301). Berlin, Heidelberg: Springer.