A New Exact Algorithm to Optimize a Linear Function Over the Set of Efficient Solutions for Bi-objective Mixed Integer Linear Programs

Alvaro Sierra-Altamiranda
Department of Industrial and Management System Engineering, University of South Florida, Tampa, FL, 33620 USA, amsierra@mail.usf.edu, http://www.eng.usf.edu/∼amsierra/

Hadi Charkhgard
Department of Industrial and Management System Engineering, University of South Florida, Tampa, FL, 33620 USA, hcharkhgard@usf.edu, http://www.eng.usf.edu/∼hcharkhgard/

We present the first (criterion space search) algorithm for optimizing a linear function over the set of efficient solutions of bi-objective mixed integer linear programs. The proposed algorithm is developed based on the Triangle Splitting Method (Boland et al. 2015b) which can find a full representation of the nondominated frontier of any bi-objective mixed integer linear program. The proposed algorithm is easy to implement and converges quickly to an optimal solution. An extensive computational study shows the efficacy of the algorithm. We numerically show that the proposed algorithm can be used to quickly generate a provably high-quality approximate solution because it maintains a lower bound and an upper bound on the optimal value of the linear function at any point in time.

Key words: bi-objective mixed integer linear programming, criterion space search algorithm, optimization over efficient frontier, triangle splitting method

History:

1. Introduction

Many real-world optimization problems involve multiple competing objectives, i.e., it is impossible to find a feasible solution that simultaneously optimizes all objectives. Consequently, it is not surprising that the focus of the multi-objective optimization community has been primarily on developing effective techniques for generating some (if not all) *efficient* solutions, i.e., solutions in which it is impossible to improve the value of one objective without a deterioration in the value of at least one other objective, of multi-objective optimization problems. This is mainly because understanding the trade-offs between objectives can help decision makers select their preferred solutions. Interested readers may refer to Dächert et al. (2012), Dächert and Klamroth (2014), Kirlik and Sayın (2014), Özpeynirci and Köksalan (2010), Lokman and

Köksalan (2013), Özlen et al. (2013), Przybylski and Gandibleux (2017), Przybylski et al. (2010), Soylu and Yıldız (2016), and Boland et al. (2015a,b, 2016a,c) for recent advances on exact solution approaches for multi-objective optimization problems.

Although understanding the trade-offs between objectives can be valuable, some researchers argue (see for instance Jorge (2009)) that presenting too many efficient solutions can confuse a decision maker, and so may make selecting a preferred solution almost impossible. An approach that alleviates this issue is finding a preferred solution among the set of efficient solutions directly and is known as *optimizing over the efficient set*, which is a global optimization problem (Benson (1984)). In this approach, the goal is to directly optimize a function (which can be linear or nonlinear) over the set of efficient solutions (preferably without enumerating all of them).

Note that there is a strong connection between the so-called *lexicographic* optimization (Ehrgott 2005) and the problem of optimizing over the efficient set. In the simplest form of lexicographic optimization, there are two functions, denoted by $z_1(\boldsymbol{x})$ and $f(\boldsymbol{x})$, that one of them has a higher priority, e.g., $z_1(\boldsymbol{x})$, compared to the other one. So, in this case, decision makers would like to find a solution that optimizes $f(\boldsymbol{x})$ among all solutions that optimizes $z_1(\boldsymbol{x})$. Now, consider a generalized version of the lexicographic optimization in which there is set of objective functions $\{z_1(\boldsymbol{x}), \ldots, z_p(\boldsymbol{x})\}$ that are equally important and they all have a higher priority compared to another objective function $f(\boldsymbol{x})$. So, in this case, decision makers would like to find a solution that optimizes $f(\boldsymbol{x})$ among all solutions that optimizes $\{z_1(\boldsymbol{x}), \ldots, z_p(\boldsymbol{x})\}$. Observe that this is precisely the problem of optimization over the efficient set. In fact, such lexicographic optimization problems arise in some real-world settings. For example, in the field of conservation planning, $z_i(\boldsymbol{x})$ can be considered as the extinction risk of species $i \in \{1, \ldots, p\}$ and $f(\boldsymbol{x})$ can be considered as the total cost (Nicholson and Possingham 2006).

Note too that developing (effective) solution approaches for optimizing over the efficient set has two other benefits in addition to the fact that it can help decision makers find their perfected solutions:

1. It can help us solve one of the classical problems in the field of multi-objective optimization: computing the *nadir point*, i.e., the point in criterion space given by the worst value of each of the objective functions over the set of efficient solutions. Obviously, to compute each component of the nadir point, a special case of the problem of optimizing over the efficient set can be solved (Boland et al. 2016b, Köksalan and Lokman 2015, Kirlik and Sayın 2015, Özpeynirci 2017).

2. It may help us solve certain single-objective optimization problems effectively. In other words, there are certain single-objective optimization problems that can be viewed as the problem of optimizing over the efficient set and can be solved faster in practice (Charkhgard et al. 2018, Gao et al. 2006, Shao and Ehrgott 2016). For example, $\max_{x\in\mathcal{X}} \prod_{i=1}^{p} z_i(x)$ when $z_i(x) \geq 0$ for all $x \in \mathcal{X}$ and $i = 1, \ldots, p$ is equivalent to the problem of maximizing $\prod_{i=1}^{p} z_i(x)$ over the set of efficient solutions of $\max_{x\in\mathcal{X}}\{z_1(x), \ldots, z_p(x)\}$.

It is worth mentioning there exist many studies about solving the problem of optimizing a linear function over the efficient set of multi-objective linear programs, see for instance Benson (1984), Benson (1991), Benson (1992), Benson (1993), Dauer (1991), Ecker and Song (1994), Sayin (2000) and Yamamoto (2002). Moreover, there are a few studies on optimizing a linear function over the set of efficient solutions of multi-objective pure integer linear programs, see for instance Abbas and Chaabane (2006), Jorge (2009), Djamal and Marc (2010), Chaabane et al. (2012) and Boland et al. (2016b).

However, no exact algorithm is known for optimizing a linear function over the set of efficient solutions of multi-objective mixed integer linear programs (with an arbitrary number of objective functions). One reason for this observation is that the existing algorithms developed for pure integer cases cannot straightforwardly be applied to mixed integer cases in general. Such algorithms use the fact that the nondominated frontier of pure integer instances is discrete. So, there exists an $\varepsilon > 0$ that can capture the minimum distance between any two points in the nondominated frontier of such problems. For example, if all coefficients of the objective functions are integers then ε can be safely set to 1. For mixed integer instances the only accurate value for ε is 0. However, those algorithms by construction never terminate if one chooses a value of 0 or close to it for ε. This is because they keep producing some nondominated points again and again if ε is very small. Also, if one chooses a large value for ε then the algorithm terminates but it will not be an exact method anymore. Because such an algorithm basically approximates the nondominated frontier by a finite set of discrete points and so they conduct the optimization over the approximate efficient set.

Another reason is that, to this date, no exact algorithm has been developed for finding a full representation of *the nondominated frontier*, i.e., the set of points in criterion space corresponding to the set of efficient solutions, of multi-objective mixed integer linear programs with an arbitrary number of objective functions. If such an algorithm exists then they can be possibly modified for optimizing a linear function over the set of efficient solutions. However, fortunately, since a few algorithms have been recently developed for finding a full

representation of the nondominated frontier of bi-objective mixed integer linear programs (see for instance Boland et al. (2015b), Fattahi and Turkay (2018), Soylu and Yıldız (2016), and Vincent et al. (2013)), generating an exact algorithm for optimizing a linear function over the efficient set of bi-objective mixed integer linear programs should now be possible.

In light of the above, the main contribution of our research is to develop the first algorithm for optimizing a linear function over the set of efficient solutions of bi-objective mixed integer linear programs. Our algorithm employs the main components of the triangle splitting method which is one of the fastest exact *criterion space search algorithms*, i.e., methods that search in the space of objective function values, for computing a full representation of the nondominated frontier of a bi-objective mixed integer linear programs (Boland et al. 2015b). By conducting an extensive computational study, we demonstrate the efficacy of the proposed algorithm. The new algorithm has the following desirable characteristics:

- It is both easy to understand and easy to implement.
- It maintains a global lower bound and a global upper bound for the linear function at any point in time. So the algorithm is inherently good for quickly generating a provably high-quality approximate solution.
- The algorithm has minimal requirements in terms of information storage.
- The algorithm relies on solving single-objective mixed integer linear programs, and so it automatically benefits from any advances on single-objective mixed integer linear programming solvers.

The rest of this paper is organized as follows. In Section 2, we explain the main concepts in bi-objective mixed integer linear programming and also provide a high-level description of the triangle splitting method and some necessary mathematical operations. In Section 3, we describe the problem of optimizing over the efficient set and sketch the basis of our proposed algorithm. In Section 4, we present our proposed algorithm in detail. In Section 5, we graphically show the performance of the proposed algorithm using a small example. In Section 6, we describe some implementation issues and enhancement techniques. In Section 7, we provide a comprehensive computational study. Finally, in Section 8, we provide some concluding remarks.

2. Preliminaries

In this section, we provide some preliminaries that are essential for explaining the contents of other sections.

2.1. Bi-objective mixed integer linear programming

A *Bi-Objective Mixed Integer Linear Program* (BOMILP) can be stated as follows:

$$\min_{(\boldsymbol{x}_I,\boldsymbol{x}_C)\in\mathcal{X}} \{z_1(\boldsymbol{x}_I,\boldsymbol{x}_C), z_2(\boldsymbol{x}_I,\boldsymbol{x}_C)\}, \tag{1}$$

where $\mathcal{X} := \left\{(\boldsymbol{x}_I,\boldsymbol{x}_C) \in \mathbb{Z}^{n_1}_{\geq} \times \mathbb{R}^{n_2}_{\geq} : A_1\boldsymbol{x}_I + A_2\boldsymbol{x}_C \leq \boldsymbol{b}\right\}$ represents the *feasible set in the decision space*, $\mathbb{Z}^{n_1}_{\geq} := \{\boldsymbol{s} \in \mathbb{Z}^{n_1} : \boldsymbol{s} \geq \boldsymbol{0}\}$, $\mathbb{R}^{n_2}_{\geq} := \{\boldsymbol{s} \in \mathbb{R}^{n_2} : \boldsymbol{s} \geq \boldsymbol{0}\}$, $A_1 \in \mathbb{R}^{m\times n_1}$, $A_2 \in \mathbb{R}^{m\times n_2}$, and $\boldsymbol{b} \in \mathbb{R}^m$. It is assumed that $\mathcal{X}$ is *bounded* and $z_i(\boldsymbol{x}_I,\boldsymbol{x}_C) = \boldsymbol{c}^{\intercal}_{i,I}\boldsymbol{x}_I + \boldsymbol{c}^{\intercal}_{i,C}\boldsymbol{x}_C$ where $\boldsymbol{c}_{i,I} \in \mathbb{R}^{n_1}$ and $\boldsymbol{c}_{i,C} \in \mathbb{R}^{n_2}$ for $i = 1, 2$ represents a linear objective function. The image $\mathcal{Y}$ of $\mathcal{X}$ under vector-valued function $\boldsymbol{z} := (z_1, z_2)^{\intercal}$ represents the *feasible set in the objective/criterion space*, that is $\mathcal{Y} := \{\boldsymbol{o} \in \mathbb{R}^2 : \boldsymbol{o} = \boldsymbol{z}(\boldsymbol{x}_I,\boldsymbol{x}_C)$ for all $(\boldsymbol{x}_I,\boldsymbol{x}_C) \in \mathcal{X}\}$.

Definition 1. A feasible solution $(\boldsymbol{x}_I,\boldsymbol{x}_C) \in \mathcal{X}$ is called *efficient* or, if there is no other $(\boldsymbol{x}'_I,\boldsymbol{x}'_C) \in \mathcal{X}$ such that $z_1(\boldsymbol{x}'_I,\boldsymbol{x}'_C) \leq z_1(\boldsymbol{x}_I,\boldsymbol{x}_C)$ and $z_2(\boldsymbol{x}'_I,\boldsymbol{x}'_C) < z_2(\boldsymbol{x}_I,\boldsymbol{x}_C)$ or $z_1(\boldsymbol{x}'_I,\boldsymbol{x}'_C) < z_1(\boldsymbol{x}_I,\boldsymbol{x}_C)$ and $z_2(\boldsymbol{x}'_I,\boldsymbol{x}'_C) \leq z_2(\boldsymbol{x}_I,\boldsymbol{x}_C)$. If $(\boldsymbol{x}_I,\boldsymbol{x}_C)$ is efficient, then $\boldsymbol{z}(\boldsymbol{x}_I,\boldsymbol{x}_C)$ is called a *nondominated point*. The set of all efficient solutions is denoted by $\mathcal{X}_E$. The set of all nondominated points $\boldsymbol{z}(\boldsymbol{x}_I,\boldsymbol{x}_C)$ for $(\boldsymbol{x}_I,\boldsymbol{x}_C) \in \mathcal{X}_E$ is denoted by $\mathcal{Y}_N$ and referred to as the *nondominated frontier*.

Definition 2. If there exists a vector $(\lambda_1,\lambda_2)^{\intercal} \in \mathbb{R}^2_{>} := \{\boldsymbol{s} \in \mathbb{R}^2 : \boldsymbol{s} > \boldsymbol{0}\}$ such that $(\boldsymbol{x}^*_I,\boldsymbol{x}^*_C) \in \arg\min_{(\boldsymbol{x}_I,\boldsymbol{x}_C)\in\mathcal{X}} \lambda_1 z_1(\boldsymbol{x}_I,\boldsymbol{x}_C) + \lambda_2 z_2(\boldsymbol{x}_I,\boldsymbol{x}_C)$, then $(\boldsymbol{x}^*_I,\boldsymbol{x}^*_C)$ is called a *supported efficient solution* and $\boldsymbol{z}(\boldsymbol{x}^*_I,\boldsymbol{x}^*_C)$ is called a *supported nondominated point*.

Definition 3. Let $\mathcal{Y}^e$ be the set of extreme points of the convex hull of $\mathcal{Y}$, that is the smallest convex set containing the set $\mathcal{Y}$. A point $\boldsymbol{z}(\boldsymbol{x}_I,\boldsymbol{x}_C) \in \mathcal{Y}$ is called an *extreme supported nondominated point*, if $\boldsymbol{z}(\boldsymbol{x}_I,\boldsymbol{x}_C)$ is a supported nondominated point and $\boldsymbol{z}(\boldsymbol{x}_I,\boldsymbol{x}_C) \in \mathcal{Y}^e$.

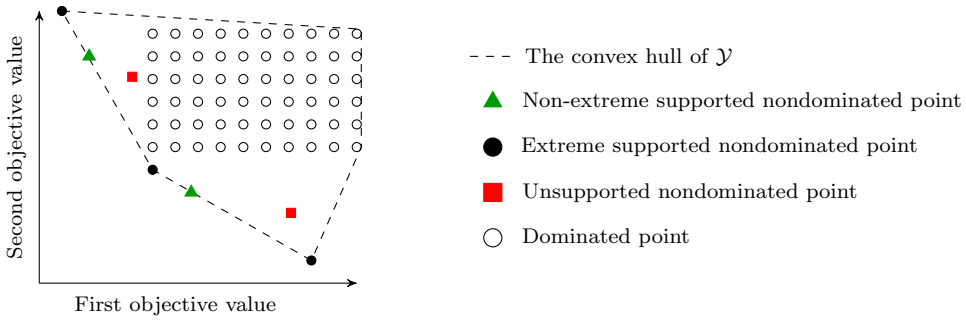


Figure 1 **An illustration of different types of (feasible) points in the criterion space**

In summary, based on Definition 1, the elements of $\mathcal{Y}$ can be divided into two groups including dominated and nondominated points. Furthermore, based on Definitions 2 and 3,

the nondominated points can be divided into unsupported nondominated points, non-extreme supported nondominated points and extreme supported nondominated points. Overall, bi-objective optimization problems are concerned with finding an exact representation of the elements of $\mathcal{Y}_N$, i.e., all nondominated points including supported and unsupported nondominated points. An illustration of the set $\mathcal{Y}$ when $n_2 = 0$, i.e., there is no continuous variable, and its corresponding categories are shown in Figure 1.

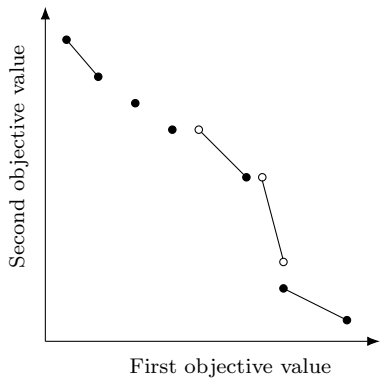


Figure 2 An example of the nondominated frontier of a BOMILP

Note that if $n_1 > 0$ and $n_2 > 0$, the nondominated frontier of a BOMILP can be more complicated since some continuous segments may appear in the nondominated frontier. A typical example of a nondominated frontier of a BOMILP is shown in Figure 2. Observe that isolated points as well as closed, half-open, and open segments may exist in the nondominated frontier. So, although we assumed that $\mathcal{X}$ is bounded, we observe that the nondominated frontier of a BOMILP may still contain an infinite number of nondominated points (because of the existence of continuous segments) and so computing a (finite) exact representation of the nondominated frontier is quite challenging. So, that is one of the main reasons that only very few exact algorithms are developed for BOMILPs.

In general, another important difficulty of dealing with BOMILPs that (to the best of our knowledge) is not discussed enough in the literature is the implementation issue. Developing a stable prototype implementation of the exact algorithms for BOMILPs is not an easy task and the literature is suffering from that. When it comes to the implementation, different existing algorithms may perform differently on a given instance. Although they can approximately generate the nondominated frontier with (possibly) high and acceptable level of precision, they may be different in terms of outcomes (for a fixed level of precision). For example, some may report more nondominated line segments or points, and some may report less. However, finding

more points and segments does not necessarily imply superiority because numerical issues can occur all the time when dealing with mixed integer instances, especially for instances that their nondominated frontier is almost continuous and has a high level of curvature. In light of this observation, we believe that the C++ prototype implementation of the algorithm that we have developed in this study is stable. This is due to the fact that we have also implemented the same algorithm in other programming languages and they all report the same.

2.2. Optimization over the efficient set of a BOMILP

The problem of optimizing a linear function over the set of efficient solutions of a BOMILP can be stated as follows:

$$\min_{(\boldsymbol{x}_I,\boldsymbol{x}_C)\in\mathcal{X}_E} \quad f(\boldsymbol{x}_I,\boldsymbol{x}_C) \tag{2}$$

where $\mathcal{X}_E$ is the set of efficient solutions of Problem (1) and $f(\boldsymbol{x}_I,\boldsymbol{x}_C)=\boldsymbol{c}_{f,I}^{\mathsf{T}}\boldsymbol{x}_I+\boldsymbol{c}_{f,C}^{\mathsf{T}}\boldsymbol{x}_C$ where $\boldsymbol{c}_{f,I}\in\mathbb{R}^{n_1}$ and $\boldsymbol{c}_{f,C}\in\mathbb{R}^{n_2}$ represents a linear function. To ensure that the problem cannot be solved straightforwardly, we assume that $\mathcal{X}\neq\mathcal{X}_E$ and $f(\boldsymbol{x})$ is not a strictly positive linear combination of $z_1(\boldsymbol{x})$ and $z_2(\boldsymbol{x})$. Note that if $\mathcal{X}=\mathcal{X}_E$ or $f(\boldsymbol{x})$ is a strictly positive linear combination of $z_1(\boldsymbol{x})$ and $z_2(\boldsymbol{x})$ then Problem (2) is equivalent to $\min_{(\boldsymbol{x}_I,\boldsymbol{x}_C)\in\mathcal{X}} f(\boldsymbol{x}_I,\boldsymbol{x}_C)$.

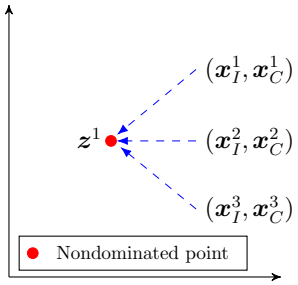


Figure 3 **Image of some efficient solutions in the criterion space**

Next, we provide some notation and operations that form the basis for the new algorithm. However, before that we make two important observations.

OBSERVATION 1. There may exist one or more efficient solutions with the same image in the criterion space. So, after discovering a nondominated point, we often need to find a solution among all feasible solutions corresponding to that point which has the minimum value for $f(\boldsymbol{x}_I,\boldsymbol{x}_C)$. For example, in Figure 3, it is observed that $(\boldsymbol{x}_I^1,\boldsymbol{x}_C^1)$, $(\boldsymbol{x}_I^2,\boldsymbol{x}_C^2)$ and $(\boldsymbol{x}_I^3,\boldsymbol{x}_C^3)$ return the same nondominated point. Now, if we have that $f(\boldsymbol{x}_I^1,\boldsymbol{x}_C^1)<f(\boldsymbol{x}_I^2,\boldsymbol{x}_C^2)<f(\boldsymbol{x}_I^3,\boldsymbol{x}_C^3)$ then $(\boldsymbol{x}_I^1,\boldsymbol{x}_C^1)$ provides the best upper (or primal) bound for the optimal value of Problem (2).

OBSERVATION 2. Let $f^l:=\min_{(\boldsymbol{x}_I,\boldsymbol{x}_C)\in\mathcal{X}} \quad f(\boldsymbol{x}_I,\boldsymbol{x}_C)$ and $f^*=\min_{(\boldsymbol{x}_I,\boldsymbol{x}_C)\in\mathcal{X}_E} \quad f(\boldsymbol{x}_I,\boldsymbol{x}_C)$. We must have that $f^l\leq f^*$, i.e., f^l is a lower (or dual) bound.

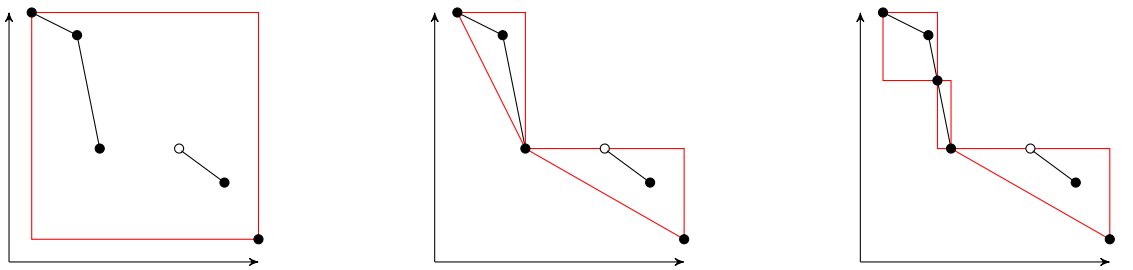
Overall, we use the notation $(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l) \in \mathcal{X}$ for a local lower bound solution and $(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u) \in \mathcal{X}_E$ for a local upper bound solution.

3. Key Operations

In this section, we first present a brief explanation of the Triangle Splitting Method (TSM) and its key operations that form the basis of our algorithm (Boland et al. 2015b). We also describe some operations that do not exist in TSM but are necessary for our algorithm to compute efficient solutions and bounds.

3.1. The triangle splitting method

One of the effective (criterion space search) algorithms for BOMILPs is TSM (Boland et al. 2015b). Since this algorithm provides the basis for the development of our algorithm, we next explain a high-level description of TSM. However, before doing so, we note that Boland et al. (2015b) has developed an optional and heuristic operation for simplifying the results that TSM reports. So, after producing the nondominated frontier by TSM, users can optionally call this operation to obtain a simpler representation of the frontier with the cost of losing the accuracy. So, such an operation is not required and so will not be employed in this study.



(a) Finding endpoints (b) Generated triangles (c) Generated rectangles

Figure 4 Progression of TSM in terms of the discovery of nondominated points

TSM maintains a list of rectangles and right-angled triangles (in the criterion space) that need to be explored. At the beginning, this list is empty. So, TSM first computes the endpoints of the nondominated frontier. These two points are then used to define the first rectangle containing all the "not yet found" nondominated points, as shown in Figure 4a. The algorithm explores a rectangle by finding the locally extreme supported nondominated points within the rectangle. Now it can be shown that by finding these points, the rectangle can be split into a set of right-angled triangles containing all "not yet found" nondominated points, as shown in Figure 4b. The algorithm explores a triangle by first checking whether its hypotenuse is part of the nondominated frontier. If that is the case, then the triangle is removed from

the list, otherwise it is split into at most two other rectangles. This operation is further illustrated in Figure 4c, where the hypotenuse of the top triangle in Figure 4b is not part of the nondominated frontier and so it is split into two new rectangles. The algorithm repeats these procedures until finding a full representation of the nondominated frontier, i.e., the list of rectangles and triangles becomes empty.

In lights of the above, we now explain the notation and operations that are used in TSM and are necessary for the development of our algorithm. Let $\boldsymbol{z}^1 = (z_1^1, z_2^1)$ and $\boldsymbol{z}^2 = (z_1^2, z_2^2)$ be two points in the criterion space with $z_1^1 \leq z_1^2$ and $z_2^1 \geq z_2^2$. We denote by $R(\boldsymbol{z}^1, \boldsymbol{z}^2)$ the rectangle in the criterion space defined by the points $\boldsymbol{z}^1$ and $\boldsymbol{z}^2$. Furthermore, we denote by $T(\boldsymbol{z}^1, \boldsymbol{z}^2)$ the right-angled triangle in the criterion space defined by the points $\boldsymbol{z}^1$, (z_1^2, z_2^1), and $\boldsymbol{z}^2$. Finally, we denote by $H(\boldsymbol{z}^1, \boldsymbol{z}^2)$ the line segment in the criterion space defined by the points $\boldsymbol{z}^1$ and $\boldsymbol{z}^2$, i.e., the hypotenuse of triangle $T(\boldsymbol{z}^1, \boldsymbol{z}^2)$.

Lexicographic Operation. The top endpoint, denoted by $\boldsymbol{z}^T$, of the nondominated frontier can be found by solving two (single-objective) Mixed Integer Linear Programs (MILPs) in sequence, as follows:

$$z_1^T = \min_{(\boldsymbol{x}_I, \boldsymbol{x}_C) \in \mathcal{X}} z_1(\boldsymbol{x}_I, \boldsymbol{x}_C)$$

(if feasible) followed by

$$z_2^T = \min_{(\boldsymbol{x}_I, \boldsymbol{x}_C) \in \mathcal{X}} \{z_2(\boldsymbol{x}_I, \boldsymbol{x}_C) : z_1(\boldsymbol{x}_I, \boldsymbol{x}_C) \leq z_1^T\}.$$

As this operation will be called frequently in our algorithm, we introduce the following notation to represent the process:

$$\boldsymbol{z}^T = \operatorname*{lex\,min}_{(\boldsymbol{x}_I, \boldsymbol{x}_C) \in \mathcal{X}} \{z_1(\boldsymbol{x}_I, \boldsymbol{x}_C), z_2(\boldsymbol{x}_I, \boldsymbol{x}_C)\},$$

Similarly, the bottom endpoint, denoted by $\boldsymbol{z}^B$, of the nondominated frontier can be found by solving:

$$\boldsymbol{z}^B = \operatorname*{lex\,min}_{(\boldsymbol{x}_I, \boldsymbol{x}_C) \in \mathcal{X}} \{z_2(\boldsymbol{x}_I, \boldsymbol{x}_C), z_1(\boldsymbol{x}_I, \boldsymbol{x}_C)\}.$$

Weighted Sum Method Operation. Another frequently used operation is the weighted sum method (Aneja and Nair (1979)). We denote this operation by Weighted-Sum-Method$(\boldsymbol{z}^1, \boldsymbol{z}^2)$ where $\boldsymbol{z}^1, \boldsymbol{z}^2 \in \mathcal{Y}_N$. This operation uses the following optimization problem to find all locally extreme supported nondominated points of a given rectangle $R(\boldsymbol{z}^1, \boldsymbol{z}^2)$:

$$\begin{aligned} \boldsymbol{z}^* = \min_{(\boldsymbol{x}_I, \boldsymbol{x}_C) \in \mathcal{X}} \quad & \lambda_1 z_1(\boldsymbol{x}_I, \boldsymbol{x}_C) + \lambda_2 z_2(\boldsymbol{x}_I, \boldsymbol{x}_C) \\ \text{subject to} \quad & \boldsymbol{z}(\boldsymbol{x}_I, \boldsymbol{x}_C) \in R(\boldsymbol{z}^1, \boldsymbol{z}^2), \end{aligned}$$

where $\lambda_1, \lambda_2 > 0$. We note that only the value of λ_1 and λ_2 will be iteratively updated during the course of the weighted sum method (and not the set of constraints). In other words, in each iteration, for a given pair of nondominated points $(\boldsymbol{z}', \boldsymbol{z}'')$ with $z'_1 < z''_1$ and $z'_2 > z''_2$, we set $\lambda_1 = z'_2 - z''_2$ and $\lambda_2 = z''_1 - z'_1$. The first pair of points that will be used is $(\boldsymbol{z}^1, \boldsymbol{z}^2)$. It is not hard to see that using this pair, the value of objective function is parallel to the line that connects the points $\boldsymbol{z}^1$ and $\boldsymbol{z}^2$ in the criterion space as shown in Figure 5. We note that by Definition 2, the corresponding optimization problem always returns a new locally supported nondominated point $\boldsymbol{z}^*$. However, this point can be an extreme point only if $\lambda_1 z^*_1 + \lambda_2 z^*_2 < \lambda_1 z^1_1 + \lambda_2 z^1_2$. So, if this condition holds, the process will be repeated recursively for pairs $(\boldsymbol{z}^1, \boldsymbol{z}^*)$ and $(\boldsymbol{z}^*, \boldsymbol{z}^2)$. Overall, at the end of the weighted sum method, the list of points that we have discovered contains all locally extreme supported nondominated points as well as (possibly) some non-extreme supported nondominated points.

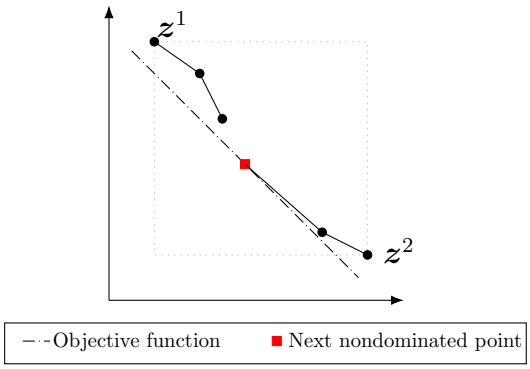


Figure 5 Weighted sum optimization problem

Line Detector Operation. This operation is a (single-objective) MILP that determines whether some or all points on the hypotenuse of a given triangle $T(\boldsymbol{z}^1, \boldsymbol{z}^2)$ are nondominated points based on the following theorem, and it is denoted by Line-Detector$(\boldsymbol{z}^1, \boldsymbol{z}^2)$.

THEOREM 1. *Let $\boldsymbol{z}^1$ and $\boldsymbol{z}^2$ be two nondominated points in the criterion space. If $(H(\boldsymbol{z}^1, \boldsymbol{z}^2) - \mathbb{R}^2_>) \cap \mathcal{Y}_N = \emptyset$ and there exists an $\boldsymbol{x}_I \in \mathcal{X}_I$ and $\boldsymbol{x}^1_C$ and $\boldsymbol{x}^2_C \in \mathbb{R}^m$ such that $(\boldsymbol{x}_I, \boldsymbol{x}^1_C), (\boldsymbol{x}_I, \boldsymbol{x}^2_C) \in \mathcal{X}$, $z_1(\boldsymbol{x}_I, \boldsymbol{x}^1_C) \leq z^1_1$, $z_2(\boldsymbol{x}_I, \boldsymbol{x}^1_C) \leq z^1_2$, $z_1(\boldsymbol{x}_I, \boldsymbol{x}^2_C) \leq z^2_1$, and $z_2(\boldsymbol{x}_I, \boldsymbol{x}^2_C) \leq z^2_2$, then $H(\boldsymbol{z}^1, \boldsymbol{z}^2) \subseteq \mathcal{Y}_N$.* (Boland et al. 2015b)

Specifically, suppose that for a given triangle $T(\boldsymbol{z}^1, \boldsymbol{z}^2)$ generated during the course of TSM, we simultaneously find two feasible solutions $(\boldsymbol{x}_I, \boldsymbol{x}^1_C), (\boldsymbol{x}_I, \boldsymbol{x}^2_C) \in \mathcal{X}$ such that the following three conditions hold:

- If there is a difference between solutions, it is only because of the values of continuous decision variables.
- For the first solution, we must have that $\boldsymbol{z}(\boldsymbol{x}_I, \boldsymbol{x}_C^1) = \boldsymbol{z}^1$.
- For the second solution, we must have that $\boldsymbol{z}(\boldsymbol{x}_I, \boldsymbol{x}_C^2) \in H(\boldsymbol{z}^1, \boldsymbol{z}^2)$.

If these conditions hold at the same time then the line segment between the point $\boldsymbol{z}^1$ and $\boldsymbol{z}(\boldsymbol{x}_I, \boldsymbol{x}_C^2)$ is part of the nondominated frontier. Obviously, such a line segment is a subset of $H(\boldsymbol{z}^1, \boldsymbol{z}^2)$, so the goal would be to maximize the length of the line segment to hopefully cover the entire hypotenuse. This can be achieved by solving the following MILP:

$$\begin{aligned} \max \ & z_1(\boldsymbol{x}_I, \boldsymbol{x}_C^2) \\ \text{subject to} \ & z_1(\boldsymbol{x}_I, \boldsymbol{x}_C^1) \leq z_1^1 \\ & z_2(\boldsymbol{x}_I, \boldsymbol{x}_C^1) \leq z_2^1 \\ & \lambda_1 z_1(\boldsymbol{x}_I, \boldsymbol{x}_C^2) + \lambda_2 z_2(\boldsymbol{x}_I, \boldsymbol{x}_C^2) = \lambda_1 z_1^1 + \lambda_2 z_2^1 \\ & (\boldsymbol{x}_I, \boldsymbol{x}_C^1) \in \mathcal{X}, (\boldsymbol{x}_I, \boldsymbol{x}_C^2) \in \mathcal{X} \end{aligned}$$

where $\lambda_1 = z_2^1 - z_2^2$ and $\lambda_2 = z_1^2 - z_1^1$. Note that the first and second constraints can be written in the form of equality as well but since $\boldsymbol{z}^1$ is a nondominated point it is computationally better to use inequalities. An illustration of this optimization problem can be found in Figure 6.

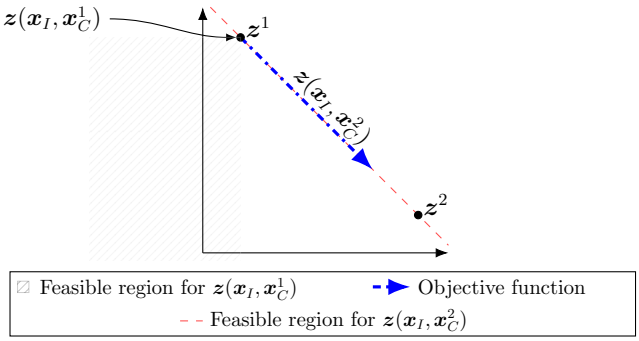


Figure 6 **Line detector operation**

Triangle Split Operation. This operation splits a given triangle $T(\boldsymbol{z}^1, \boldsymbol{z}^2)$. We denote this operation by Split-Triangle$(\boldsymbol{z}^1, \boldsymbol{z}^2, horizontal)$ or Split-Triangle$(\boldsymbol{z}^1, \boldsymbol{z}^2, vertical)$ when we split a triangle first horizontally (and then possibly vertically) or first vertically (and then possibly horizontally), respectively. We now explain Split-Triangle$(\boldsymbol{z}^1, \boldsymbol{z}^2, horizontal)$ graphically as shown in Figure 7.

In this operation, as shown in Figure 7a, we first cut the triangle into two parts horizontally and then explore the bottom part of the triangle to compute the top (local) endpoint of the

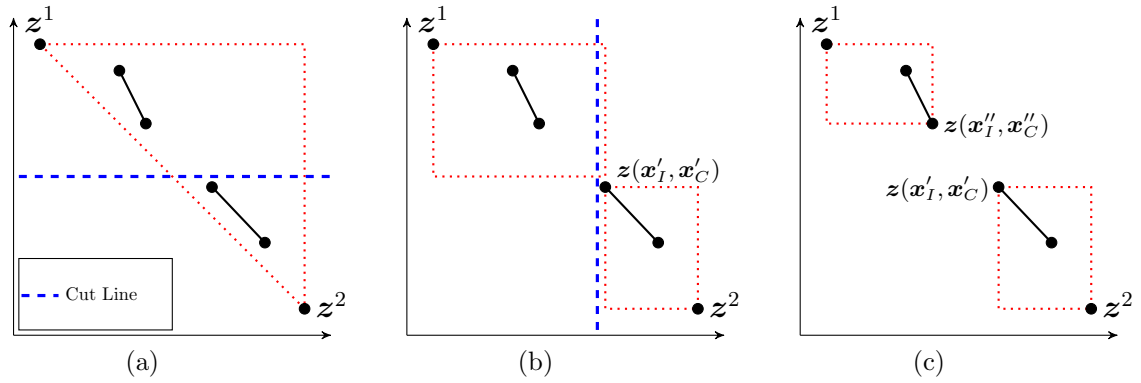


Figure 7 **Horizontal splitting of triangle $\mathbf{T}(z^1, z^2)$**

nondominated frontier in the search region. This can be done using the following lexicographic operation:

$$(\boldsymbol{x}'_I, \boldsymbol{x}'_C) \in \underset{(\boldsymbol{x}_I, \boldsymbol{x}_C) \in \mathcal{X}}{\arg \operatorname{lex} \min} \{z_1(\boldsymbol{x}_I, \boldsymbol{x}_C), z_2(\boldsymbol{x}_I, \boldsymbol{x}_C) : z_2(\boldsymbol{x}_I, \boldsymbol{x}_C) \leq \frac{z_2^1 + z_2^2}{2}, \boldsymbol{z}(\boldsymbol{x}_I, \boldsymbol{x}_C) \in T(\boldsymbol{z}^1, \boldsymbol{z}^2)\},$$

Next, as shown in Figure 7b, we cut the triangle vertically based on the position of $\boldsymbol{z}(\boldsymbol{x}'_I, \boldsymbol{x}'_C)$ and explore the left part of the triangle to compute the bottom (local) endpoint of the nondominated frontier in that search region. This can be done using the following lexicographic operation:

$$(\boldsymbol{x}''_I, \boldsymbol{x}''_C) \in \underset{(\boldsymbol{x}_I, \boldsymbol{x}_C) \in \mathcal{X}}{\arg \operatorname{lex} \min} \{z_2(\boldsymbol{x}_I, \boldsymbol{x}_C), z_1(\boldsymbol{x}_I, \boldsymbol{x}_C) : z_1(\boldsymbol{x}_I, \boldsymbol{x}_C) < z_1(\boldsymbol{x}'_I, \boldsymbol{x}'_C), \boldsymbol{z}(\boldsymbol{x}_I, \boldsymbol{x}_C) \in T(\boldsymbol{z}^1, \boldsymbol{z}^2)\}.$$

Finally, as shown in Figure 7c, after computing these two new nondominated points, the triangle can be changed to (at most) two new rectangles $R\big(\boldsymbol{z}^1, \boldsymbol{z}(\boldsymbol{x}''_I, \boldsymbol{x}''_C)\big)$ and $R\big(\boldsymbol{z}(\boldsymbol{x}'_I, \boldsymbol{x}'_C), \boldsymbol{z}^2\big)$. Obviously, there is no need to consider $R\big(\boldsymbol{z}^1, \boldsymbol{z}(\boldsymbol{x}''_I, \boldsymbol{x}''_C)\big)$ for further investigation if $\boldsymbol{z}^1 = \boldsymbol{z}(\boldsymbol{x}''_I, \boldsymbol{x}''_C)$. Similarly, there is no need to consider $R\big(\boldsymbol{z}(\boldsymbol{x}'_I, \boldsymbol{x}'_C), \boldsymbol{z}^2\big)$ for further investigation if $\boldsymbol{z}(\boldsymbol{x}'_I, \boldsymbol{x}'_C) = \boldsymbol{z}^2$. As an aside, to avoid spending the valuable computational time on (potential) redundant calculations, we set $(\boldsymbol{x}''_I, \boldsymbol{x}''_C) = (\boldsymbol{x}'_I, \boldsymbol{x}'_C)$ and skip solving the second lexicographic operation if the image of $(\boldsymbol{x}'_I, \boldsymbol{x}'_C)$ in the criterion space is on the horizontal cut line, i.e., $z_2(\boldsymbol{x}'_I, \boldsymbol{x}'_C) = \frac{z_2^1 + z_2^2}{2}$.

We note that Split-Triangle$(\boldsymbol{z}^1, \boldsymbol{z}^2, \textit{vertical})$ can be defined similarly. In other words, we first need to compute:

$$(\boldsymbol{x}''_I, \boldsymbol{x}''_C) \in \underset{(\boldsymbol{x}_I, \boldsymbol{x}_C) \in \mathcal{X}}{\arg \operatorname{lex} \min} \{z_2(\boldsymbol{x}_I, \boldsymbol{x}_C), z_1(\boldsymbol{x}_I, \boldsymbol{x}_C) : z_1(\boldsymbol{x}_I, \boldsymbol{x}_C) \leq \frac{z_1^1 + z_1^2}{2}, \boldsymbol{z}(\boldsymbol{x}_I, \boldsymbol{x}_C) \in T(\boldsymbol{z}^1, \boldsymbol{z}^2)\}.$$

and then follow it by:

$$(\boldsymbol{x}'_I, \boldsymbol{x}'_C) \in \underset{(\boldsymbol{x}_I, \boldsymbol{x}_C) \in \mathcal{X}}{\arg \operatorname{lex} \min} \{z_1(\boldsymbol{x}_I, \boldsymbol{x}_C), z_2(\boldsymbol{x}_I, \boldsymbol{x}_C) : z_2(\boldsymbol{x}_I, \boldsymbol{x}_C) < z_2(\boldsymbol{x}''_I, \boldsymbol{x}''_C), \boldsymbol{z}(\boldsymbol{x}_I, \boldsymbol{x}_C) \in T(\boldsymbol{z}^1, \boldsymbol{z}^2)\}.$$

It is worth mentioning that TSM (and our proposed algorithm in this study) alternates between Split-Triangle($\boldsymbol{z}^1, \boldsymbol{z}^2, horizontal$) and Split-Triangle($\boldsymbol{z}^1, \boldsymbol{z}^2, vertical$) to be able to discover both horizontal and vertical gaps in the nondominated frontier.

3.2. Computing local upper bound solutions

During the course of the new algorithm, e.g., when splitting a triangle, we often discover a nondominated point $\boldsymbol{z}^1 \in \mathcal{Y}_N$. In that case, based on Observation 1, we solve the following optimization problem to obtain a local upper bound solution:

$$(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u) \in \underset{x \in \mathcal{X}}{\arg\min} \{f(\boldsymbol{x}) : \boldsymbol{z}(\boldsymbol{x}) \leq \boldsymbol{z}^1\}.$$

We denote this operation by UB-Finder-Point($\boldsymbol{z}^1$).

When we are exploring a right-angeled triangle $T(\boldsymbol{z}^1, \boldsymbol{z}^2)$ during the course of the proposed algorithm, we may discover that the hypotenuse of the triangle is part of the nondominated frontier, i.e., $H(\boldsymbol{z}^1, \boldsymbol{z}^2) \in \mathcal{Y}_N$. So, in this case, based on Observation 1, we solve the following optimization problem to obtain a local upper bound solution:

$$(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u) \in \underset{(\boldsymbol{x}_I, \boldsymbol{x}_C) \in \mathcal{X}}{\arg\min} \{f(\boldsymbol{x}_I, \boldsymbol{x}_C) : \boldsymbol{z}(\boldsymbol{x}_I, \boldsymbol{x}_C) \in H(\boldsymbol{z}^1, \boldsymbol{z}^2)\}.$$

We denote this operation by UB-Finder-Line($\boldsymbol{z}^1, \boldsymbol{z}^2$).

3.3. Computing local lower bound solutions

Before exploring a rectangle $R(\boldsymbol{z}^1, \boldsymbol{z}^2)$ during the course of the proposed algorithm, based on Observation 2, we solve the following optimization problem to obtain a local lower bound solution:

$$(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l) \in \underset{(\boldsymbol{x}_I, \boldsymbol{x}_C) \in \mathcal{X}}{\arg\min} \{f(\boldsymbol{x}_I, \boldsymbol{x}_C) : \boldsymbol{z}(\boldsymbol{x}_I, \boldsymbol{x}_C) \in R(\boldsymbol{z}^1, \boldsymbol{z}^2)\}.$$

We denote this operation by LB-Finder-Rectangle($\boldsymbol{z}^1, \boldsymbol{z}^2$). Similarly, before exploring a triangle $T(\boldsymbol{z}^1, \boldsymbol{z}^2)$ during the course of the proposed algorithm, based on Observation 2, we solve the following optimization problem to obtain a local lower bound solution:

$$(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l) \in \underset{x \in \mathcal{X}}{\arg\min} \{f(\boldsymbol{x}_I, \boldsymbol{x}_C) : \boldsymbol{z}(\boldsymbol{x}_I, \boldsymbol{x}_C) \in T(\boldsymbol{z}^1, \boldsymbol{z}^2)\}.$$

We denote this operation by LB-Finder-Triangle($\boldsymbol{z}^1, \boldsymbol{z}^2$).

3.4. Computing an efficient solution based on a local lower solution

After computing a local lower bound solution $(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)$ by using LB-Finder-Triangle$(\boldsymbol{z}^1, \boldsymbol{z}^2)$, we often try to compute an efficient solution in $T(\boldsymbol{z}^1, \boldsymbol{z}^2)$ where $\boldsymbol{z}^1, \boldsymbol{z}^2 \in \mathcal{Y}_N$, based on $(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)$ (see Algorithm 2). In order to do so, we solve the following optimization problem:

$$(\boldsymbol{x}_I', \boldsymbol{x}_C') \in \underset{(\boldsymbol{x}_I, \boldsymbol{x}_C) \in \mathcal{X}}{\arg\min} \{z_1(\boldsymbol{x}) + z_2(\boldsymbol{x}) : \boldsymbol{z}(\boldsymbol{x}_I, \boldsymbol{x}_C) \leq \boldsymbol{z}(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)\}.$$

We denote this operation by Find-NDP$(\boldsymbol{z}(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l))$. Note that showing that this operation returns an efficient solution within the search region defined by $T(\boldsymbol{z}^1, \boldsymbol{z}^2)$ where $\boldsymbol{z}^1, \boldsymbol{z}^2 \in \mathcal{Y}_N$ is straightforward, and so we have omitted the proof. Note too that computing $(\boldsymbol{x}_I', \boldsymbol{x}_C')$ is useful since we often use it as an alternative to split $T(\boldsymbol{z}^1, \boldsymbol{z}^2)$. This implies that after computing $(\boldsymbol{x}_I', \boldsymbol{x}_C')$, we often immediately change $T(\boldsymbol{z}^1, \boldsymbol{z}^2)$ into two new rectangles, i.e., $R(\boldsymbol{z}^1, \boldsymbol{z}(\boldsymbol{x}_I', \boldsymbol{x}_C'))$ and $R(\boldsymbol{z}(\boldsymbol{x}_I', \boldsymbol{x}_C'), \boldsymbol{z}^2)$.

4. The new algorithm

The new algorithm maintains a priority queue of rectangles and triangles. Specifically, each element of the priority queue is denoted by $(\boldsymbol{z}^1, \boldsymbol{z}^2, shape, direction, LB)$ where 'shape' is either triangle or rectangle; '$\boldsymbol{z}^1$' and '$\boldsymbol{z}^2$' are the corner points of the corresponding rectangle or triangle; 'direction' is either horizontal or vertical and it indicates that if the shape is a triangle and we want to split it then how this process should be done; and 'LB' is the (local) lower bound corresponding to this element (obtained by using the techniques developed in Section 3.3).

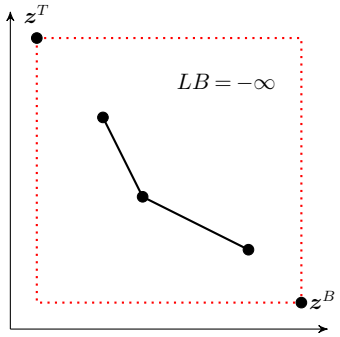


Figure 8 Initial rectangle

At the beginning of the algorithm, the priority queue is initialized with $(\boldsymbol{z}^T, \boldsymbol{z}^B, rectangle, horizontal, -\infty)$. An illustration of the initial rectangle can be found in Figure 8. The algorithm also maintains a global upper bound and global lower bound for

Problem (2) denoted by GUB and GLB, respectively. Let $(\boldsymbol{x}_I^*, \boldsymbol{x}_C^*) \in \mathcal{X}_E$ be the best efficient solution that has been found at any point during the course of the algorithm. Obviously, $GUB = f(\boldsymbol{x}_I^*, \boldsymbol{x}_C^*)$. Note that we set $GUB = +\infty$ at the beginning of the algorithm. Also, the GLB is equal to the minimum value of LB in all elements of the priority queue. We assume that the priority queue is sorted in nondecreasing order of the value of LB at any point in time. Consequently, the local lower bound corresponding to the first element of the list is always GLB. It is worth mentioning that the algorithm terminates as soon as the priority queue is empty or the optimality gap is below a certain threshold. Note that if the priority queue becomes empty then the optimality gap is naturally zero. Next, we explain how the algorithm works in each iteration.

In each iteration, the algorithm pops out the first element of the priority queue and denote it by $(\boldsymbol{z}^1, \boldsymbol{z}^2, shape, direction, LB)$. Note that when an element is popped out from the priority queue then that element does not exist in the priority queue anymore. The algorithm first sets $GLB = LB$ and then computes both the relative and absolute optimality gaps. Let $\epsilon_1, \epsilon_2 > 0$ be small positive values that are defined by users. If $\frac{|GUB - GLB|}{|GUB| + \varepsilon_1} \leq \varepsilon_2$ or $GUB \leq GLB + \varepsilon_2$ then the algorithm terminates and reports $(\boldsymbol{x}_I^*, \boldsymbol{x}_C^*)$ as an optimal solution. Note that ε_1 is just introduced to modify the denominator of the relative gap for cases with $GUB = 0$. So, in this study, we simply assume that $\varepsilon_1 = 10^{-5}$.

So, in the remaining, we assume that $\frac{|GUB - GLB|}{|GUB| + \varepsilon_1} > \varepsilon_2$ and $GUB > GLB + \varepsilon_2$. In this case, the algorithm first checks whether $\boldsymbol{z}^1 = \boldsymbol{z}^2$. If that is the case then the algorithm calls UB-Finder-Point($\boldsymbol{z}^1$) to compute an upper bound solution $(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$ and then the new iteration starts after updating GUB and $(\boldsymbol{x}_I^*, \boldsymbol{x}_C^*)$. Note that, from now on, updating GUB and $(\boldsymbol{x}_I^*, \boldsymbol{x}_C^*)$ means that if $GUB > f(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$ then we set $GUB = f(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$ and $(\boldsymbol{x}_I^*, \boldsymbol{x}_C^*) = (\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$.

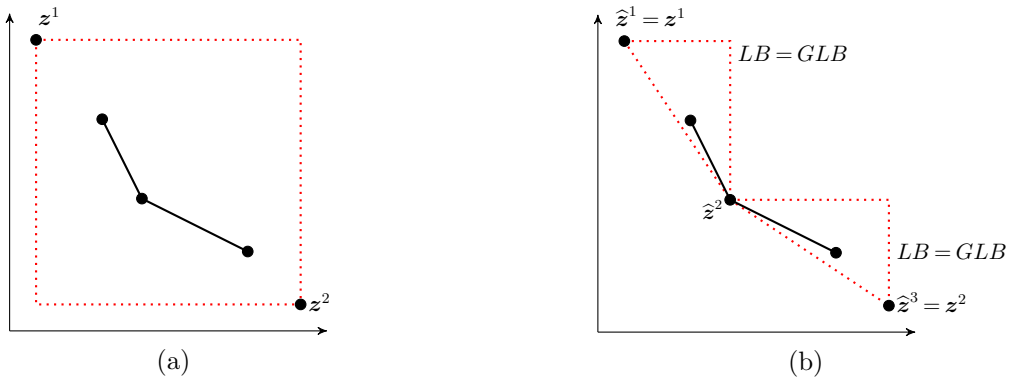


Figure 9 **Changing rectangle $R(z^1, z^2)$ into smaller triangles**

So, in the remaining, we assume that $\boldsymbol{z}^1 \neq \boldsymbol{z}^2$. Now, if the shape is rectangle then the algorithm simply applies the weighted sum method in the rectangle to find all locally extreme supported nondominated points. We denote the outcome of the weighted sum operation by $\widehat{\boldsymbol{z}}^1, \widehat{\boldsymbol{z}}^2, \ldots, \widehat{\boldsymbol{z}}^k$ such that $\widehat{z}_1^v < \widehat{z}_1^{v+1}$ and $\widehat{z}_2^v > \widehat{z}_2^{v+1}$ for each $v = 1, \ldots, k-1$. After finding these points, the algorithm adds $k-1$ new elements to the priority queue that are all triangles. Specifically, it adds $(\widehat{\boldsymbol{z}}^v, \widehat{\boldsymbol{z}}^{v+1}, \mathit{triangle}, \mathit{direction}, GLB)$ to the priority queue for each $v = 1, \ldots, k-1$. Note that the direction and the local lower bound of the new elements are the same as the direction and the local lower bound of the rectangle that is explored (we know that, by construction, $GLB = LB$). It is worth mentioning that the algorithm does not attempt to find better lower bounds for the new triangles simply because we have observed that this is computationally expensive (in practice). An illustration of the generated triangles after exploring $R(\boldsymbol{z}^1, \boldsymbol{z}^2)$ can be found in Figure 9.

Now, if the shape is triangle, i.e., $T(\boldsymbol{z}^1, \boldsymbol{z}^2)$, then the algorithm checks whether $H(\boldsymbol{z}^1, \boldsymbol{z}^2)$ is partially or entirely part of the nondominated frontier. In other words, the algorithm calls Line-Detector$(\boldsymbol{z}^1, \boldsymbol{z}^2)$ and denotes its corresponding optimal solution by $(\widehat{\boldsymbol{x}}_I, \widehat{\boldsymbol{x}}_C)$. From the theory of the line detector operation (presented in Section 3.1), we know that the line segment between $\boldsymbol{z}^1$ and $(\widehat{\boldsymbol{x}}_I, \widehat{\boldsymbol{x}}_C)$ should be part of the nondominated frontier. So, now one of the two possible cases may arise:

- **Case I:** $\boldsymbol{z}^1 \neq \boldsymbol{z}(\widehat{\boldsymbol{x}}_I, \widehat{\boldsymbol{x}}_C)$ (which is equivalent to $z_1(\widehat{\boldsymbol{x}}_I, \widehat{\boldsymbol{x}}_C) > z_1^1$ based on the construction of the line detector operation)
- **Case II:** $\boldsymbol{z}^1 = \boldsymbol{z}(\widehat{\boldsymbol{x}}_I, \widehat{\boldsymbol{x}}_C)$ (which is equivalent to $z_1(\widehat{\boldsymbol{x}}_I, \widehat{\boldsymbol{x}}_C) = z_1^1$ based on the construction of the line detector operation)

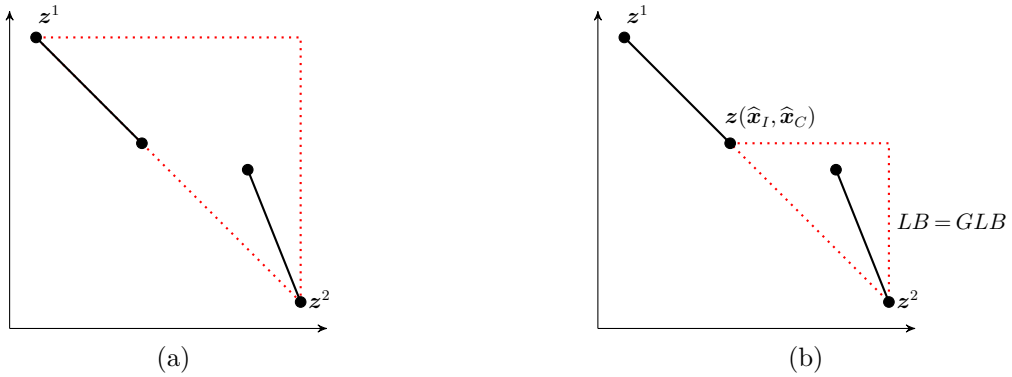


Figure 10 **Generating a smaller triangle after observing that the hypotenuse of is partially part of the nondominated frontier**

If Case I arises then the algorithm calls UB-Finder-Line $\left(\boldsymbol{z}^1, \boldsymbol{z}(\widehat{\boldsymbol{x}}_I, \widehat{\boldsymbol{x}}_C)\right)$ to compute an upper bound solution $(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$ and then updates GUB and $(\boldsymbol{x}_I^*, \boldsymbol{x}_C^*)$. Finally, if the hypotenuse is

not entirely part of the nondominated frontier, i.e., $z_1(\widehat{\boldsymbol{x}}_I, \widehat{\boldsymbol{x}}_C) < z_1^2$, then the algorithm adds $(\boldsymbol{z}(\widehat{\boldsymbol{x}}_I, \widehat{\boldsymbol{x}}_C), \boldsymbol{z}^2, \mathit{triangle}, \mathit{direction}, GLB)$ to the priority. Note that the new element is a smaller version of the initial triangle, i.e., $T(\boldsymbol{z}^1, \boldsymbol{z}^2)$. An illustration of such a triangle can be found in Figure 10.

If Case II arises then the algorithm first calls LB-Finder-Triangle$(\boldsymbol{z}^1, \boldsymbol{z}^2)$ to find a lower bound solution $(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)$ within the search region defined by the triangle. Obviously, if $f(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l) \geq GUB$ then there is no need to explore the triangle any further and so the algorithm starts a new iteration. Otherwise, we need to split the triangle, i.e., $T(\boldsymbol{z}^1, \boldsymbol{z}^2)$, into two new rectangles, denoted by $R(\boldsymbol{z}^1, \underline{\boldsymbol{z}})$ and $R(\bar{\boldsymbol{z}}, \boldsymbol{z}^2)$ where $\bar{\boldsymbol{z}}, \underline{\boldsymbol{z}} \in \mathcal{Y}_N$ and $\bar{\boldsymbol{z}}, \underline{\boldsymbol{z}} \in T(\boldsymbol{z}^1, \boldsymbol{z}^2)$. We also denoted by $\underline{L}$ and $\bar{L}$ a (local) lower bound corresponding to rectangles $R(\boldsymbol{z}^1, \underline{\boldsymbol{z}})$ and $R(\bar{\boldsymbol{z}}, \boldsymbol{z}^2)$, respectively. In order to compute $\bar{\boldsymbol{z}}, \underline{\boldsymbol{z}}, \underline{L}$ and $\bar{L}$, we have developed an operation denoted by Explore-Triangle $\big((\boldsymbol{x}_I^l, \boldsymbol{x}_C^l), \boldsymbol{z}^1, \boldsymbol{z}^2, \mathit{direction}\big)$. This operation, in addition to $\bar{\boldsymbol{z}}, \underline{\boldsymbol{z}}, \underline{L}$ and $\bar{L}$, returns a new 'direction' and a new upper bound solution $(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$ as well. Later in this section, we explain this operation in detail.

In light of the above, after calling Explore-Triangle $\big((\boldsymbol{x}_I^l, \boldsymbol{x}_C^l), \boldsymbol{z}^1, \boldsymbol{z}^2, \mathit{direction}\big)$, the algorithm first updates GUB and $(\boldsymbol{x}_I^*, \boldsymbol{x}_C^*)$ using the new upper bound solution $(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$. The algorithm then adds the element $(\boldsymbol{z}^1, \underline{\boldsymbol{z}}, \mathit{rectangle}, \mathit{direction}, \underline{L})$ to the priority queue if $\boldsymbol{z}^1 \neq \underline{\boldsymbol{z}}$ (since otherwise the shape is a single point and not a rectangle) and $\underline{L} < GUB$ (since otherwise such a rectangle cannot contain a better solution). Similarly, the algorithm adds $(\bar{\boldsymbol{z}}, \boldsymbol{z}^2, \mathit{rectangle}, \mathit{direction}, \bar{L})$ to the priority queue if $\boldsymbol{z}^2 \neq \bar{\boldsymbol{z}}$ and $\bar{L} < GUB$.

A detailed description of the proposed algorithm can be found in Algorithm 1.

4.1. Operation Explore-Triangle

We now explain the details of the operation Explore-Triangle $\big((\boldsymbol{x}_I^l, \boldsymbol{x}_C^l), \boldsymbol{z}^1, \boldsymbol{z}^2, \mathit{direction}\big)$. Evidently, the inputs of this operation are $(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)$, $\boldsymbol{z}^1$, $\boldsymbol{z}^2$ and *direction*. The outputs of this operation are $(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u), (\underline{\boldsymbol{z}}, \underline{L}), (\bar{\boldsymbol{z}}, \bar{L})$ and *direction*.

As discussed previously, the operation provides the necessary components for splitting the right-angeled triangle $T(\boldsymbol{z}^1, \boldsymbol{z}^2)$ into (at most) two new rectangles. This operation first checks whether the image of $(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)$ in the criterion space is (reasonably) far away from the orthogonal sides of the right-angeled triangle $T(\boldsymbol{z}^1, \boldsymbol{z}^2)$ (the reason for checking this will become clear in the upcoming paragraphs). In other words, it checks whether $z_1(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l) < z_1^2 - (z_1^2 - z_1^1)\varepsilon_3$ and $z_2(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l) < z_2^1 - (z_2^1 - z_2^2)\varepsilon_3$ where $\epsilon_3 \in (0, 1)$ is a user defined parameter. The default value of ϵ_3 is 0.15 in our implementation of the algorithm since this value performs the best for all our computational experiments. So, two possible cases may arise:

Algorithm 1: The new algorithm

$GLB \leftarrow -\infty$; $GUB \leftarrow +\infty$; $PQ.create(P)$; $PQ.add(P, (\boldsymbol{z}^T, \boldsymbol{z}^B, rectangle, horizontal, -\infty))$
$Search\_Done \leftarrow False$
while *not* $PQ.empty(P)$ **and** $Search\_Done = False$ **do**
  $PQ.pop(P, (\boldsymbol{z}^1, \boldsymbol{z}^2, shape, direction, LB))$
  $GLB \leftarrow LB$
  if $\frac{|GUB-GLB|}{|GUB|+\varepsilon_1} \leq \varepsilon_2$ **or** $GUB \leq GLB + \varepsilon_2$ **then**
    $Search\_Done \leftarrow True$
  else
    if $\boldsymbol{z}^1 = \boldsymbol{z}^2$ **then**
      $(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u) \leftarrow$ UB-Finder-Point$(\boldsymbol{z}^1)$
      if $f(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u) < GUB$ **then**
        $GUB \leftarrow f(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$
        $(\boldsymbol{x}_I^*, \boldsymbol{x}_C^*) \leftarrow (\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$
    else
      if $shape = rectangle$ **then**
        $\{\widehat{\boldsymbol{z}}^1, \widehat{\boldsymbol{z}}^2, ..., \widehat{\boldsymbol{z}}^k\} \leftarrow$ Weighted-Sum-Method$(\boldsymbol{z}^1, \boldsymbol{z}^2)$
        $i \leftarrow 1$
        while $i < k$ **do**
          $PQ.add(P, (\widehat{\boldsymbol{z}}^i, \widehat{\boldsymbol{z}}^{i+1}, triangle, direction, GLB)$
          $i \leftarrow i + 1$
      else
        $(\widehat{\boldsymbol{x}}_I, \widehat{\boldsymbol{x}}_C) \leftarrow$ Line-Detector$(\boldsymbol{z}^1, \boldsymbol{z}^2)$
        if $z_1(\widehat{\boldsymbol{x}}_I, \widehat{\boldsymbol{x}}_C) > z_1^1$ **then**
          $(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u) \leftarrow$ UB-Finder-Line $\left(\boldsymbol{z}^1, \boldsymbol{z}(\widehat{\boldsymbol{x}}_I, \widehat{\boldsymbol{x}}_C)\right)$
          if $f(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u) < GUB$ **then**
            $GUB \leftarrow f(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$
            $(\boldsymbol{x}_I^*, \boldsymbol{x}_C^*) \leftarrow (\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$
          if $z_1(\widehat{\boldsymbol{x}}_I, \widehat{\boldsymbol{x}}_C) < z_1^2$ **then**
            $PQ.add(P, (\boldsymbol{z}(\widehat{\boldsymbol{x}}_I, \widehat{\boldsymbol{x}}_C), \boldsymbol{z}^2, triangle, direction, GLB))$
        else
          $(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l) \leftarrow$ LB-Finder-Triangle$(\boldsymbol{z}^1, \boldsymbol{z}^2)$
          if $f(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l) < GUB$ **then**
            $\left((\boldsymbol{x}_I^u, \boldsymbol{x}_C^u), (\underline{\boldsymbol{z}}, \underline{L}), (\bar{\boldsymbol{z}}, \bar{L}), direction\right) \leftarrow$ Explore-Triangle $\left((\boldsymbol{x}_I^l, \boldsymbol{x}_C^l), \boldsymbol{z}^1, \boldsymbol{z}^2, direction\right)$
            if $f(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u) < GUB$ **then**
              $GUB \leftarrow f(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$
              $(\boldsymbol{x}_I^*, \boldsymbol{x}_C^*) \leftarrow (\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$
            if $\underline{L} < GUB$ **and** $\boldsymbol{z}^1 \neq \underline{\boldsymbol{z}}$ **then**
              $PQ.add(P, (\boldsymbol{z}^1, \underline{\boldsymbol{z}}, rectangle, direction, \underline{L}))$
            if $\bar{L} < GUB$ **and** $\boldsymbol{z}^2 \neq \bar{\boldsymbol{z}}$ **then**
              $PQ.add(P, (\bar{\boldsymbol{z}}, \boldsymbol{z}^2, rectangle, direction, \bar{L}))$
return $(\boldsymbol{x}_I^*, \boldsymbol{x}_C^*)$ and $f(\boldsymbol{x}_I^*, \boldsymbol{x}_C^*)$

- **Case A:** $z_1(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l) < z_1^2 - (z_1^2 - z_1^1)\varepsilon_3$ and $z_2(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l) < z_2^1 - (z_2^1 - z_2^2)\varepsilon_3$.
- **Case B:** $z_1(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l) \geq z_1^2 - (z_1^2 - z_1^1)\varepsilon_3$ or $z_2(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l) \geq z_2^1 - (z_2^1 - z_2^2)\varepsilon_3$ (or both).

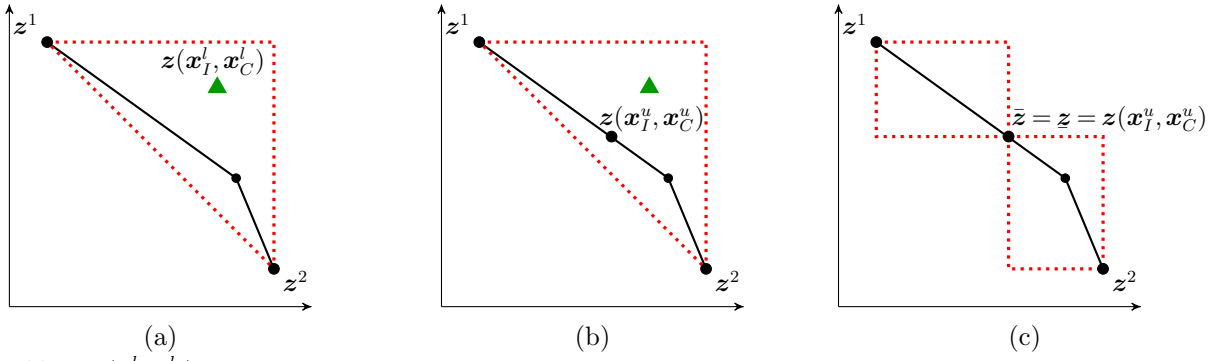


Figure 11 $\boldsymbol{z}(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)$ **is far away from the orthogonal sides of the triangle**

We first explain Case A. An illustration of this case is shown in Figure 11. For Case A, the operation first calls Find-NDP $\left(\boldsymbol{z}(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)\right)$ to find an efficient solution based on $(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)$. We denote this efficient solution by $(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$ since $f(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$ is an upper bound for the optimal objective value of Problem (2). An illustration of $(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$ in the criterion space can be found in Figure 11b. Obviously, $\boldsymbol{z}(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u) \in T(\boldsymbol{z}^1, \boldsymbol{z}^2)$. However, we must have that $\boldsymbol{z}(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u) \neq \boldsymbol{z}^1$ and $\boldsymbol{z}(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u) \neq \boldsymbol{z}^2$ and this is exactly the consequence of the fact that $z_1(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l) < z_1^2 - (z_1^2 - z_1^1)\varepsilon_3$ and $z_2(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l) < z_2^1 - (z_2^1 - z_2^2)\varepsilon_3$. The operation then sets $\bar{\boldsymbol{z}} = \underline{\boldsymbol{z}} = \boldsymbol{z}(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$. Note that this implies that Algorithm 1 will later split the triangle from the point $\boldsymbol{z}(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$ (see Figure 11c). So, there is no need to split the triangle in the standard way that TSM employs (see Section 3.1).

Finally, we note that the operation also needs to compute lower bounds (for the new rectangles). Obviously, if $\boldsymbol{z}(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u) \neq \boldsymbol{z}(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)$ then the operation immediately calls LB-Finder-Rectangle $(\boldsymbol{z}^1, \underline{\boldsymbol{z}})$ and LB-Finder-Rectangle $(\bar{\boldsymbol{z}}, \boldsymbol{z}^2)$ to compute $\underline{L}$ and $\bar{L}$, respectively. Otherwise, there is no need to call them since by definition $(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)$ is a lower bound solution which is happened to be an efficient solution as well. So, there is no need to split the triangle for further investigation because $(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)$ is an optimal solution of Problem (2) when its search is restricted to the search region defined by $T(\boldsymbol{z}^1, \boldsymbol{z}^1)$. Consequently, in this case, the operation simply sets $\bar{L} = f(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)$, $\underline{L} = f(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)$ and $(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u) = (\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)$. Note that by doing so, Algorithm 1 will not add new rectangles to the priority queue (see Lines 34-40).

We now explain Case B. In this case, the operation attempts to split the triangle using the standard way that TSM employs (see Section 3.1). This implies that if $direction = horizontal$ then the operation calls Split-Triangle$(\boldsymbol{z}^1, \boldsymbol{z}^2, horizontal)$ to compute $(\boldsymbol{x}_I', \boldsymbol{x}_C')$ and $(\boldsymbol{x}_I'', \boldsymbol{x}_C'')$,

and then it changes the *direction* to 'vertical' (since as we explained in Section 3.1, the direction should alternate in TSM). Similarly, if $direction = vertical$ then the operation calls Split-Triangle$(\boldsymbol{z}^1, \boldsymbol{z}^2, vertical)$ to compute $(\boldsymbol{x}'_I, \boldsymbol{x}'_C)$ and $(\boldsymbol{x}''_I, \boldsymbol{x}''_C)$, and then it changes the *direction* to 'horizontal'.

Note that $(\boldsymbol{x}'_I, \boldsymbol{x}'_C)$ and $(\boldsymbol{x}''_I, \boldsymbol{x}''_C)$ are efficient solutions. So, next the operation calls UB-Finder-Point $\big(\boldsymbol{z}(\boldsymbol{x}'_I, \boldsymbol{x}'_C)\big)$ to compute an upper solution denoted by $(\bar{\boldsymbol{x}}_I, \bar{\boldsymbol{x}}_C)$. Also, it calls UB-Finder-Point $\big(\boldsymbol{z}(\boldsymbol{x}''_I, \boldsymbol{x}''_C)\big)$to compute an upper solution denoted by $(\underline{\boldsymbol{x}}_I, \underline{\boldsymbol{x}}_C)$ if $\boldsymbol{z}(\boldsymbol{x}'_I, \boldsymbol{x}'_C) \neq \boldsymbol{z}(\boldsymbol{x}''_I, \boldsymbol{x}''_C)$. Otherwise, it simply sets $(\underline{\boldsymbol{x}}_I, \underline{\boldsymbol{x}}_C) = (\bar{\boldsymbol{x}}_I, \bar{\boldsymbol{x}}_C)$. After computing $(\underline{\boldsymbol{x}}_I, \underline{\boldsymbol{x}}_C)$ and $(\bar{\boldsymbol{x}}_I, \bar{\boldsymbol{x}}_C)$, the operation sets $\bar{\boldsymbol{z}} = \boldsymbol{z}(\bar{\boldsymbol{x}}_I, \bar{\boldsymbol{x}}_C)$ and $\underline{\boldsymbol{z}} = \boldsymbol{z}(\underline{\boldsymbol{x}}_I, \underline{\boldsymbol{x}}_C)$. The operation then computes the best upper bound solution as follows:

$$(\boldsymbol{x}^u_I, \boldsymbol{x}^u_C) \in \arg\min\{f(\underline{\boldsymbol{x}}_I, \underline{\boldsymbol{x}}_C), f(\bar{\boldsymbol{x}}_I, \bar{\boldsymbol{x}}_C)\}.$$

Finally, we again note that the operation also needs to compute lower bounds (for the new rectangles). If $\underline{\boldsymbol{z}} \neq \boldsymbol{z}^1$ then the operation immediately calls LB-Finder-Rectangle $(\boldsymbol{z}^1, \underline{\boldsymbol{z}})$ to compute $\underline{L}$. Otherwise, it sets $\underline{L} = f(\underline{\boldsymbol{x}}_I, \underline{\boldsymbol{x}}_C)$ since, by construction, $(\underline{\boldsymbol{x}}_I, \underline{\boldsymbol{x}}_C)$ should provide a lower bound. Similarly, if $\bar{\boldsymbol{z}} \neq \boldsymbol{z}^2$ then the operation immediately calls LB-Finder-Rectangle $(\bar{\boldsymbol{z}}, \boldsymbol{z}^2)$ to compute $\bar{L}$. Otherwise, it sets $\bar{L} = f(\bar{\boldsymbol{x}}_I, \bar{\boldsymbol{x}}_C)$.

A detailed description of Explore-Triangle $\big((\boldsymbol{x}^l_I, \boldsymbol{x}^l_C), \boldsymbol{z}^1, \boldsymbol{z}^2, direction\big)$ is shown in Algorithm 2. We now make one final comment to avoid redundant calculations:

- For Case B, before calling LB-Finder-Rectangle $(\boldsymbol{z}^1, \underline{\boldsymbol{z}})$, we first need to check whether $\boldsymbol{z}(\boldsymbol{x}^l_I, \boldsymbol{x}^l_C) \in R(\boldsymbol{z}^1, \underline{\boldsymbol{z}})$. If that is the case then we know that $\underline{L} = f(\boldsymbol{x}^l_I, \boldsymbol{x}^l_C)$. Similarly, before calling LB-Finder-Rectangle $(\bar{\boldsymbol{z}}, \boldsymbol{z}^2)$, we first need to check whether $\boldsymbol{z}(\boldsymbol{x}^l_I, \boldsymbol{x}^l_C) \in R(\bar{\boldsymbol{z}}, \boldsymbol{z}^2)$. If that is the case then we know that $\bar{L} = f(\boldsymbol{x}^l_I, \boldsymbol{x}^l_C)$.

5. A small example

In this section, we use the example introduced by Belotti et al. (2013):

$$\begin{aligned}
\max \quad & z_1(\boldsymbol{x}_I, \boldsymbol{x}_C) = -3x_1 - 6x_2 + 3x_3 + 5x_4 \\
\max \quad & z_2(\boldsymbol{x}_I, \boldsymbol{x}_C) = 15x_1 + 4x_2 + x_3 + 2x_4 \\
\text{s.t.} \quad & -x_1 + 3x_5 \leq 0 \\
& x_1 - 6x_5 \leq 0 \\
& -x_2 + 3x_5 \leq 0 \\
& x_2 - 6x_5 \leq 0
\end{aligned}$$

Algorithm 2: Explore-Triangle $\big((\boldsymbol{x}_I^l, \boldsymbol{x}_C^l), \boldsymbol{z}^1, \boldsymbol{z}^2, direction\big)$

if $z_1(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l) < z_1^2 - (z_1^2 - z_1^1)\varepsilon_3$ **and** $z_2(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l) < z_2^1 - (z_2^1 - z_2^2)\varepsilon_3$ **then**
  $(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u) \leftarrow$ Find-NDP $\big(\boldsymbol{z}(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)\big)$
  $\bar{\boldsymbol{z}} \leftarrow \boldsymbol{z}(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$; $\underline{\boldsymbol{z}} \leftarrow \boldsymbol{z}(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u)$
  if $\boldsymbol{z}(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u) = \boldsymbol{z}(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)$ **then**
    $\bar{L} \leftarrow f(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)$; $\underline{L} \leftarrow f(\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)$; $(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u) \leftarrow (\boldsymbol{x}_I^l, \boldsymbol{x}_C^l)$
  else
    $\underline{L} \leftarrow$ LB-Finder-Rectangle $(\boldsymbol{z}^1, \underline{\boldsymbol{z}})$
    $\bar{L} \leftarrow$ LB-Finder-Rectangle $(\bar{\boldsymbol{z}}, \boldsymbol{z}^2)$
else
  if $direction = horizontal$ **then**
    $\big((\boldsymbol{x}_I', \boldsymbol{x}_C'), (\boldsymbol{x}_I'', \boldsymbol{x}_C'')\big) \leftarrow$ Split-Triangle$(\boldsymbol{z}^1, \boldsymbol{z}^2, horizontal)$
    $direction \leftarrow vertical$
  else
    $\big((\boldsymbol{x}_I', \boldsymbol{x}_C'), (\boldsymbol{x}_I'', \boldsymbol{x}_C'')\big) \leftarrow$ Split-Triangle$(\boldsymbol{z}^1, \boldsymbol{z}^2, vertical)$
    $direction \leftarrow horizontal$
  $(\bar{\boldsymbol{x}}_I, \bar{\boldsymbol{x}}_C) \leftarrow$ UB-Finder-Point $\big(\boldsymbol{z}(\boldsymbol{x}_I', \boldsymbol{x}_C')\big)$
  if $\boldsymbol{z}(\boldsymbol{x}_I', \boldsymbol{x}_C') \neq \boldsymbol{z}(\boldsymbol{x}_I'', \boldsymbol{x}_C'')$ **then**
    $(\underline{\boldsymbol{x}}_I, \underline{\boldsymbol{x}}_C) \leftarrow$ UB-Finder-Point $\big(\boldsymbol{z}(\boldsymbol{x}_I'', \boldsymbol{x}_C'')\big)$
  else
    $(\underline{\boldsymbol{x}}_I, \underline{\boldsymbol{x}}_C) \leftarrow (\bar{\boldsymbol{x}}_I, \bar{\boldsymbol{x}}_C)$
  $\bar{\boldsymbol{z}} \leftarrow \boldsymbol{z}(\bar{\boldsymbol{x}}_I, \bar{\boldsymbol{x}}_C)$
  $\underline{\boldsymbol{z}} \leftarrow \boldsymbol{z}(\underline{\boldsymbol{x}}_I, \underline{\boldsymbol{x}}_C)$
  $(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u) \leftarrow \arg\min\{f(\bar{\boldsymbol{x}}_I, \bar{\boldsymbol{x}}_C), f(\underline{\boldsymbol{x}}_I, \underline{\boldsymbol{x}}_C)\}$
  if $\underline{\boldsymbol{z}} = \boldsymbol{z}^1$ **then**
    $\underline{L} \leftarrow f(\underline{\boldsymbol{x}}_I, \underline{\boldsymbol{x}}_C)$
  else
    $\underline{L} \leftarrow$ LB-Finder-Rectangle $\big(\boldsymbol{z}^1, \underline{\boldsymbol{z}}\big)$
  if $\bar{\boldsymbol{z}} = \boldsymbol{z}^2$ **then**
    $\bar{L} \leftarrow f(\bar{\boldsymbol{x}}_I, \bar{\boldsymbol{x}}_C)$
  else
    $\bar{L} \leftarrow$ LB-Finder-Rectangle $\big(\bar{\boldsymbol{z}}, \boldsymbol{z}^2\big)$
return $(\boldsymbol{x}_I^u, \boldsymbol{x}_C^u), (\underline{\boldsymbol{z}}, \underline{L}), (\bar{\boldsymbol{z}}, \bar{L}), direction$

$$
\begin{aligned}
-x_3 + 4x_6 &\leq 0 \\
x_3 - 4.5x_6 &\leq 0 \\
-x_4 + 4x_6 &\leq 0 \\
x_4 - 4.5x_6 &\leq 0 \\
x_5 + x_6 &\leq 5 \\
x_1,\ x_2,\ x_3,\ x_4 &\in \mathbb{R}_+ \\
x_5,\ x_6 &\in \mathbb{Z}_+.
\end{aligned}
$$

Note that because the proposed algorithm works with $z_1(\boldsymbol{x}_I, \boldsymbol{x}_C)$ and $z_2(\boldsymbol{x}_I, \boldsymbol{x}_C)$ in minimization form, the coefficients of the objective functions are multiplied by -1 during the course of the algorithm. However, for consistency with Belotti et al. (2013), we report their values as if we are solving the maximization form in all figures in this section.

The (exact) nondominated frontier of the presented BOMILP is shown in Figure 12. Now suppose that we want to solve $\min_{(\boldsymbol{x}_I, \boldsymbol{x}_C) \in \mathcal{X}_E} f(\boldsymbol{x}_I, \boldsymbol{x}_C)$ where $\mathcal{X}_E$ is the set of efficient solutions of the presented BOMILP. We constructed four different settings in which the only difference between them is the linear function, i.e., $f(\boldsymbol{x}_I, \boldsymbol{x}_C)$.

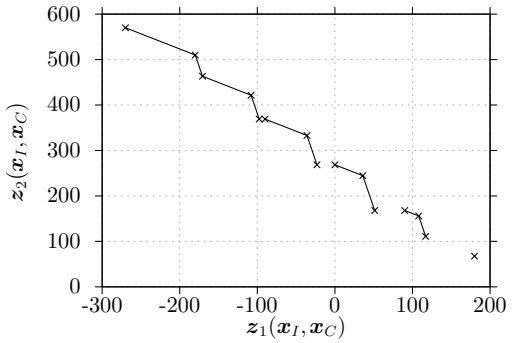


Figure 12 The exact nondominated frontier of the example

For the first three settings, the coefficients of $f(\boldsymbol{x}_I, \boldsymbol{x}_C)$ are drawn randomly from the discrete uniform distribution from the interval $[-100, 100]$ for both continuous and integer variables. For the fourth setting, we set $f(\boldsymbol{x}_I, \boldsymbol{x}_C) = z_1(\boldsymbol{x}_I, \boldsymbol{x}_C) + z_2(\boldsymbol{x}_I, \boldsymbol{x}_C)$. Note that we considered this linear function since minimizing $f(\boldsymbol{x}_I, \boldsymbol{x}_C)$ becomes contradictory to the nature of problem (in which $z_1(\boldsymbol{x}_I, \boldsymbol{x}_C)$ and $z_2(\boldsymbol{x}_I, \boldsymbol{x}_C)$ should be maximized). So, we expect that such a problem to be harder to solve. The partial nondominated frontiers generated during the course of the proposed algorithm for Settings 1 to 4 are shown in Figures 13a to 13d, respectively. In these figures, we also report $f(\boldsymbol{x}_I, \boldsymbol{x}_C)$, its optimal value (denoted by f^*), and the image of the obtained optimal solution in the criterion space. The latter is shown by a triangle symbol.

Observe that in all settings, only a proportion of the true nondominated frontier is explored during the course of algorithm. Of course, this is precisely what we hoped to achieve (in practice) by the proposed algorithm. Also, not surprisingly, for Setting 4, we see that the a larger proportion of the nondominated frontier is explored since the algorithm obtains weak lower/dual bounds when $f(\boldsymbol{x}_I, \boldsymbol{x}_C) = z_1(\boldsymbol{x}_I, \boldsymbol{x}_C) + z_2(\boldsymbol{x}_I, \boldsymbol{x}_C)$.

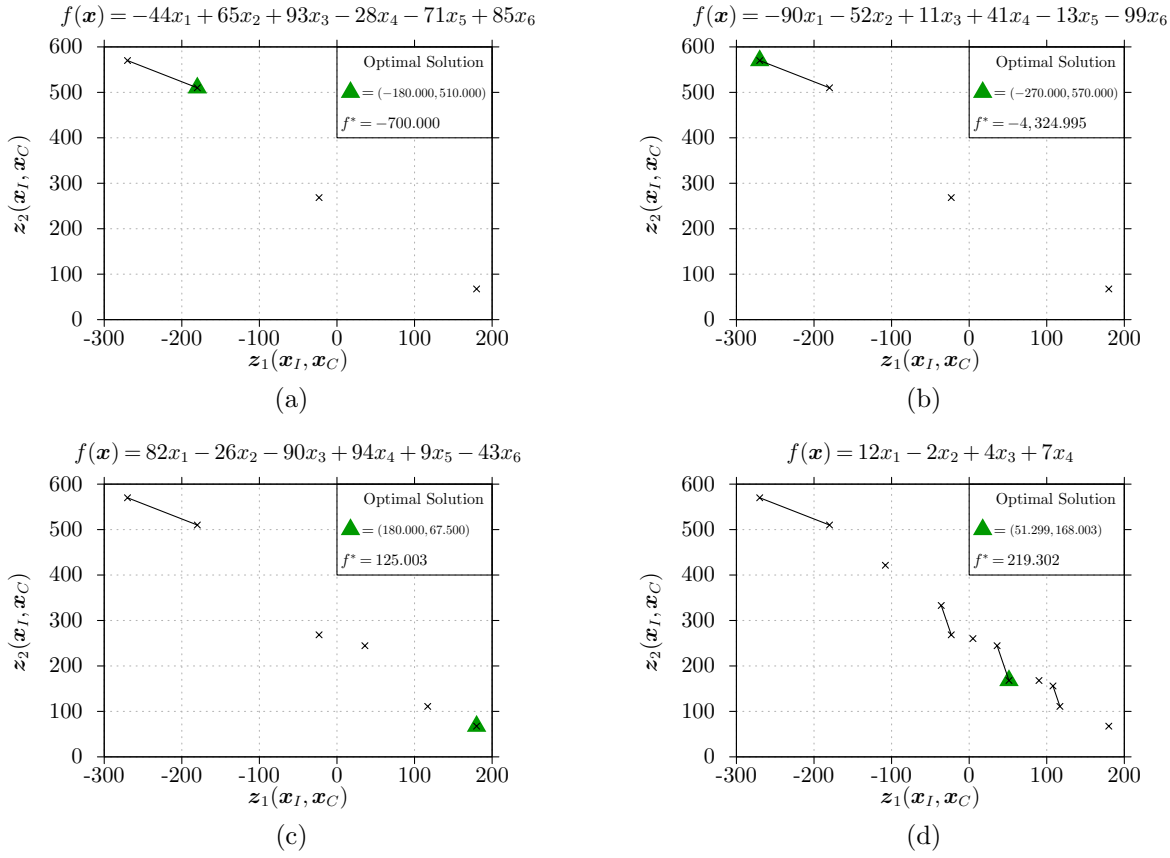


Figure 13 **The partial nondominated frontier of the example generated under each setting**

6. Implementation issues and enhancements

Since our proposed algorithm is based on TSM, the implementation tips described in Boland et al. (2015b) should be employed. We now introduce the most important ones, but interested readers may refer to Boland et al. (2015b) for further details.

- *Tip 1:* Adding as few objective function based constraints as possible to the (single-objective) MILP formulations (arising during the course of the algorithm) often reduces the numerical issues and improves the performance of the proposed algorithm in practice. So, in the proposed algorithm, we can replace any instance of $\boldsymbol{z}(\boldsymbol{x}_I, \boldsymbol{x}_C) \in R(\boldsymbol{z}^1, \boldsymbol{z}^2)$ or $\boldsymbol{z}(\boldsymbol{x}_I, \boldsymbol{x}_C) \in T(\boldsymbol{z}^1, \boldsymbol{z}^2)$ by:

$$z_1(\boldsymbol{x}_I, \boldsymbol{x}_C) \leq z_1^2$$
$$z_2(\boldsymbol{x}_I, \boldsymbol{x}_C) \leq z_2^1.$$

This is possible since we always have that $\boldsymbol{z}^1, \boldsymbol{z}^2 \in \mathcal{Y}_N$. Similarly, $\boldsymbol{z}(\boldsymbol{x}_I, \boldsymbol{x}_C) \in H(\boldsymbol{z}^1, \boldsymbol{z}^2)$ can be simplified to:

$$z_1(\boldsymbol{x}_I, \boldsymbol{x}_C) \leq z_1^2$$

$$z_2(\boldsymbol{x}_I, \boldsymbol{x}_C) \leq z_2^1$$
$$\lambda_1 z_1(\boldsymbol{x}_I, \boldsymbol{x}_C) + \lambda_2 z_2(\boldsymbol{x}_I, \boldsymbol{x}_C) \leq \lambda_1 z_1^1 + \lambda_2 z_2^1$$

- *Tip 2:* Providing (high-quality) initial feasible solutions to a MILP solver (whenever possible) often improves the performance of the proposed algorithm in practice. One of the feature of the TSM is that it naturally can provide a feasible solution for solving any (single-objective) MILP (after finding $\boldsymbol{z}^T$ and $\boldsymbol{z}^B$) that arises during its course. The same feature is also true for the proposed algorithm. While exploring $T(\boldsymbol{z}^1, \boldsymbol{z}^2)$ or $R(\boldsymbol{z}^1, \boldsymbol{z}^2)$, the solution corresponding to $\boldsymbol{z}^1$ or $\boldsymbol{z}^2$ (or both) are feasible and so they can be provided. In general, all operations defined in this study take some points in the criterion space as inputs. So, we can provide the solutions corresponding to those points to (single-objective) MILP solvers.

Next, we present some enhancements and we later in Section 7 show their effectiveness in practice.

Enhancement 1 (E_1). When calling the operations Split-Triangle($\boldsymbol{z}^1, \boldsymbol{z}^2, horizontal$) and Split-Triangle($\boldsymbol{z}^1, \boldsymbol{z}^2, vertical$), we can sometimes avoid solving some of the optimization problems.

Specifically, we know that the first step of Split-Triangle($\boldsymbol{z}^1, \boldsymbol{z}^2, horizontal$) is to solve the following lexicographic operation:

$$(\boldsymbol{x}'_I, \boldsymbol{x}'_C) \in \underset{(\boldsymbol{x}_I, \boldsymbol{x}_C) \in \mathcal{X}}{\arg \operatorname{lex\,min}} \{z_1(\boldsymbol{x}_I, \boldsymbol{x}_C), z_2(\boldsymbol{x}_I, \boldsymbol{x}_C) : z_2(\boldsymbol{x}_I, \boldsymbol{x}_C) \leq \frac{z_2^1 + z_2^2}{2}, \boldsymbol{z}(\boldsymbol{x}_I, \boldsymbol{x}_C) \in T(\boldsymbol{z}^1, \boldsymbol{z}^2)\}.$$

Note that during the course of the algorithm, we are always in the situation that $\boldsymbol{z}^1$ are $\boldsymbol{z}^2$ are nondominated points. Consequently, if after solving the first optimization problem in the above lexicographic operation, the optimal objective value is equal to z_1^2 then we do not need to solve the second optimization problem since we must have that $\boldsymbol{z}(\boldsymbol{x}'_I, \boldsymbol{x}'_C) = \boldsymbol{z}^2$.

Also, we know that the second step of Split-Triangle($\boldsymbol{z}^1, \boldsymbol{z}^2, horizontal$) is to solve the following lexicographic operation:

$$(\boldsymbol{x}''_I, \boldsymbol{x}''_C) \in \underset{(\boldsymbol{x}_I, \boldsymbol{x}_C) \in \mathcal{X}}{\arg \operatorname{lex\,min}} \{z_2(\boldsymbol{x}_I, \boldsymbol{x}_C), z_1(\boldsymbol{x}_I, \boldsymbol{x}_C) : z_1(\boldsymbol{x}_I, \boldsymbol{x}_C) < z_1(\boldsymbol{x}'_I, \boldsymbol{x}'_C), \boldsymbol{z}(\boldsymbol{x}_I, \boldsymbol{x}_C) \in T(\boldsymbol{z}^1, \boldsymbol{z}^2)\}.$$

Consequently, if after solving the first optimization problem in the above lexicographic operation, the value of the optimal objective value is equal to z_2^1 then we do not need to solve the second optimization problem since we must have that $\boldsymbol{z}(\boldsymbol{x}''_I, \boldsymbol{x}''_C) = \boldsymbol{z}^1$.

The above observations are also valid for Split-Triangle($\boldsymbol{z}^1, \boldsymbol{z}^2, vertical$).

Enhancement 2 (E_2). Observe that the intersection of any two elements (which are either triangles or rectangles) in the priority is either empty or a corner point, i.e., $\boldsymbol{z}^1$ or $\boldsymbol{z}^2$. Consequently, for a given corner point, e.g., $\boldsymbol{z}^1$, calling the operation UB-Finder-Point($\boldsymbol{z}^1$) may be redundant (if such an operation should be called at all). Consequently, before calling UB-Finder-Point($\boldsymbol{z}^1$), we propose to check whether such an operation has been called before. If the answer is "yes" then we can simply avoid calling UB-Finder-Point($\boldsymbol{z}^1$).

Enhancement 3 (E_3). Right before calling Line-Detector($\boldsymbol{z}^1, \boldsymbol{z}^2$), we propose to always check whether all integer variables in the corresponding solutions to $\boldsymbol{z}^1$ and $\boldsymbol{z}^2$ (that the algorithm maintains in the memory) have the same value. If that is the case then $H(\boldsymbol{z}^1, \boldsymbol{z}^2)$ is entirely part of the nondominated frontier. So, we can avoid solving Line-Detector($\boldsymbol{z}^1, \boldsymbol{z}^2$).

Enhancement 4 (E_4). Our numerical experiments show that Line-Detector($\boldsymbol{z}^1, \boldsymbol{z}^2$) is an expensive operation, and so it is better to postpone it. Let n_I be the number of integer variables with same values in the corresponding solutions to $\boldsymbol{z}^1$ and $\boldsymbol{z}^2$ (that the algorithm maintains in the memory). We propose to call Line-Detector($\boldsymbol{z}^1, \boldsymbol{z}^2$) only if $n_I \geq 0.9n_1$. Note that if we do not call this operation, we simply assume that $H(\boldsymbol{z}^1, \boldsymbol{z}^2)$ does not partially or entirely belong to the nondominated frontier.

7. A computational study

We conduct a comprehensive computational study to evaluate the performance of the new algorithm. We use the C++ programming language to implement the proposed approach, and employ CPLEX 12.7 as the single-objective mixed integer linear programming solver. All computational experiments are carried out on a Dell PowerEdge R630 with two Intel Xeon E5-2650 2.2 GHz 12-Core Processors (30MB), 128GB RAM, and the RedHat Enterprise Linux 6.8 operating system. We only use a single thread for all experiments.

To test the performance of the proposed algorithm, we modify 25 BOMILPs used in Boland et al. (2015b) to generate instances for Problem (2). It is worth mentioning that these 25 BOMILPs are divided into 5 classes (each contains 5 BOMILPs). Each class is denoted by Cm where m is the number of constraints of each BOMILP in that class. The number of variables and the number of constraints of each instance are equal, i.e., $(n_1 + n_2 = m)$, and half of the variables are binary while the remaining variables are continuous.

From each BOMILP, we generate 10 different instances for Problem (2). The only difference between these 10 instances is the linear function, i.e., $f(\boldsymbol{x}_I, \boldsymbol{x}_C)$. The coefficients of the linear function are randomly drawn from the discrete uniform distribution in the interval $[-100, 100]$

for both continuous and integer variables. Note that for each BOMILP, the generated linear functions can be different. Consequently, in total, $25 \times 10 = 250$ instances are tested during this comprehensive study.

The code and instances used in this study can all be found at `https://goo.gl/dixCCi` and `https://goo.gl/DzcCgQ`, respectively. It is worth mentioning that our code is generic and user-friendly in a sense that users only have to provide an input file with a specific format to use our code.

7.1. Overall performance

In this section, we show the overall performance of the proposed algorithm. We note that in all results' tables in this section, all enhancements discussed in Section 6 are active. Moreover, for each BOMILP/row, averages over the generated 10 instances are reported.

A direct way for optimizing a linear function over the set of efficient solutions of a BOMILP is to first generate the entire nondominated frontier and then applying a post-processing technique to compute an optimal solution. Note that such a post-processing technique is (of course) computationally expensive since multiple optimization problems have to be solved. In light of this observation, we first compare the performance of the our algorithm with TSM in this section. It is clear that if we show that our approach is faster than TSM then the proposed algorithm is of practical value. Note that, in this study, we do not consider the time that has to be added to the time of TSM because of the post-processing.

In Table 1, we present a comparison between the proposed algorithm and TSM where 'Time (sec.)' is the solution time in seconds and '#MILP' is the number of (single-objective) MILPs solved. The last two columns of the table show the percentage of decrease in the solution time and the number of (single-objective) MILPs solved by the proposed algorithm in comparison to TSM. Bold numbers in the last two columns show the instances that our algorithm has a better performance on them. It is also worth mentioning that the C++ implementation of TSM used in this study is exactly the one that is used in Boland et al. (2015b).

It is observed that the proposed algorithm clearly outperforms TSM in terms of solution time for most instances. Only for small size instances, the solution time of TSM is better. For large size instances, the solution time of the proposed algorithm is around 22% less than the solution time of TSM on average. Overall, in terms of the number of (single-objective) MILPs solved, the proposed algorithm is better. For the largest class of instances, we see around 11% reduction in the number of (single-objective) MILPs solved.

Table 1 Performance of the new algorithm in comparison to TSM

Class	New algorithm		TSM		% Decrease	
	Time (sec.)	#MILP	Time (sec.)	#MILP	Time	#MILP
C20	0.15	86.3	0.18	70.0	**15.0%**	-23.3%
	0.59	188.8	0.52	171.0	-12.5%	-10.4%
	0.56	170.9	0.51	151.0	-10.0%	-13.2%
	0.82	220.3	0.76	200.0	-7.5%	-10.2%
	0.17	87.1	0.18	74.0	**7.2%**	-17.7%
Avg.	0.46	150.7	0.43	133.2	-6.2%	-13.1%
C40	5.71	710.9	6.99	763.0	**18.3%**	**6.8%**
	1.49	283.9	1.98	318.0	**24.8%**	**10.7%**
	2.24	327.6	2.80	351.0	**19.9%**	**6.7%**
	2.50	397.5	2.55	392.0	**1.9%**	-1.4%
	2.58	386.5	2.35	319.0	-9.8%	-21.2%
Avg.	2.91	421.3	3.33	428.6	**12.8%**	**1.7%**
C80	41.76	1,760.5	48.41	1,700.0	**13.7%**	-3.6%
	20.98	1,028.7	31.75	1,245.0	**33.9%**	**17.4%**
	34.47	1,538.7	49.97	1,920.0	**31.0%**	**19.9%**
	45.27	1,881.2	52.01	1,841.0	**13.0%**	-2.2%
	29.40	1,480.7	35.70	1,342.0	**17.7%**	-10.3%
Avg.	34.38	1,538.0	43.57	1,609.6	**21.1%**	**4.5%**
C160	249.68	2,448.2	294.68	2,470.0	**15.3%**	**0.9%**
	256.88	2,143.8	340.96	2,334.0	**24.7%**	**8.1%**
	234.42	2,197.5	312.30	2,260.0	**24.9%**	**2.8%**
	549.73	4,299.2	766.72	4,593.0	**28.3%**	**6.4%**
	267.27	2,521.5	324.97	2,541.0	**17.8%**	**0.8%**
Avg.	311.60	2,722.0	407.93	2,839.6	**23.6%**	**4.1%**
C320	3,621.63	4,041.5	4,594.85	4,490.0	**21.2%**	**10.0%**
	5,790.73	6,341.2	6,448.02	6,404.0	**10.2%**	**1.0%**
	3,992.90	4,448.0	5,519.19	5,167.0	**27.7%**	**13.9%**
	4,757.52	4,967.3	6,652.65	6,016.0	**28.5%**	**17.4%**
	3,449.56	4,117.5	4,611.47	4,865.0	**25.2%**	**15.4%**
Avg.	4,322.47	4,783.1	5,565.24	5,388.4	**22.3%**	**11.2%**

Table 2 provides more details about the main operations of the new algorithm, i.e., Algorithm 1. We observe that Line-Detector operation takes the most time to solve a (single-objective) MILP with the overall average of $0.77\frac{\text{sec.}}{\#\text{MILPs}}$. That is the main reason that we develop enhancements E_3 and E_4 in Section 6. We also note that the second most expensive operation is the Explore-Triangle with the overall average of $0.65\frac{\text{sec.}}{\#\text{MILPs}}$.

7.2. Performance enhancements

In Section 6, we described a total of four enhancements to improve the performance of the proposed algorithm. Let 'time ratio' of A (for a particular instance) be the ratio of the solution time of A (for that instance) to the maximum solution time of the new algorithm observed during the entire set of experiments (for that instance). Figure 14 shows a comparison between time ratios (for all 250 instances) when a particular enhancement is disabled. The first box, i.e., 'All', refers to a configuration that all enhancements are disabled. In the second box, i.e., 'None', no enhancement is disabled. Boxes labeled E_1, E_2, E_3, and E_4 refer to configurations that only enhancements E_1, E_2, E_3, and E_4 are disabled, respectively.

Table 2 Basic performance statistics of the operations of the new algorithm

	Total		Weighted-Sum-Method		Line-Detector		UB-Finder-Line		LB-Finder-Triangle		Explore-Triangle	
	Time (sec.)	#MILP	Time (sec.)	#MILP	Time (sec.)	#MILP	Time (sec.)	#MILP	Time (sec.)	#MILP	Time (sec.)	#MILP
C20	0.15	86.30	0.07	32.70	0.00	0.00	0.02	14.20	0.01	5.50	0.06	29.90
	0.59	188.80	0.23	77.00	0.00	0.00	0.10	32.80	0.04	12.00	0.22	63.00
	0.56	170.90	0.12	39.50	0.00	0.10	0.04	10.30	0.07	16.00	0.32	101.00
	0.82	220.30	0.30	88.90	0.00	0.00	0.12	38.30	0.06	14.80	0.34	74.30
	0.17	87.10	0.08	41.40	0.00	0.00	0.02	18.80	0.01	4.00	0.06	18.90
Avg.	**0.46**	**150.68**	**0.16**	**55.90**	**0.00**	**0.02**	**0.06**	**22.88**	**0.04**	**10.46**	**0.20**	**57.42**
C40	5.71	710.90	2.14	330.30	0.03	2.30	1.25	146.00	0.39	39.40	1.91	188.90
	1.49	283.90	0.57	119.60	0.00	0.00	0.22	50.70	0.14	17.30	0.57	92.30
	2.24	327.60	0.81	134.40	0.02	3.60	0.36	55.90	0.19	20.20	0.85	109.50
	2.50	397.50	1.01	198.00	0.00	0.00	0.49	89.90	0.21	18.20	0.79	87.40
	2.58	386.50	0.89	156.10	0.00	0.00	0.39	65.30	0.21	26.20	1.08	134.90
Avg.	**2.91**	**421.28**	**1.08**	**187.68**	**0.01**	**1.18**	**0.54**	**81.56**	**0.23**	**24.26**	**1.04**	**122.60**
C80	41.76	1,760.50	15.94	872.30	1.32	55.40	10.54	412.40	2.50	75.90	11.45	340.50
	20.98	1,028.70	7.74	497.50	0.59	34.00	4.65	230.90	1.32	44.30	6.68	218.00
	34.47	1,538.70	11.16	640.10	1.38	74.40	5.75	269.70	3.12	104.30	13.05	446.20
	45.27	1,881.20	15.73	871.00	1.73	68.90	10.37	403.40	3.03	95.20	14.39	438.70
	29.40	1,480.70	10.62	701.10	1.01	60.10	6.32	331.20	2.06	67.60	9.37	316.70
Avg.	**34.38**	**1,537.96**	**12.24**	**716.40**	**1.20**	**58.56**	**7.52**	**329.52**	**2.40**	**77.46**	**10.99**	**352.02**
C160	249.68	2,448.20	67.87	949.90	19.13	169.00	43.60	434.10	22.21	170.00	96.78	721.20
	256.88	2,143.80	70.18	842.70	20.61	141.30	44.25	384.20	23.54	149.30	98.25	622.30
	234.42	2,197.50	70.89	889.00	16.44	148.10	41.73	401.30	20.13	148.10	85.16	607.00
	549.73	4,299.20	172.91	1,742.10	47.82	266.30	104.47	816.30	41.20	275.70	183.25	1,194.80
	267.27	2,521.50	81.68	1,109.60	19.49	137.30	61.77	509.40	20.17	142.70	84.09	618.50
Avg.	**311.60**	**2,722.04**	**92.71**	**1,106.66**	**24.70**	**172.40**	**59.16**	**509.06**	**25.45**	**177.16**	**109.51**	**752.76**
C320	3,621.63	4,041.50	879.89	1,325.70	381.81	340.30	529.98	626.50	342.79	340.10	1,486.74	1,404.90
	5,790.73	6,341.20	1,401.19	2,115.90	640.13	520.80	880.67	1,014.80	518.21	519.20	2,350.11	2,166.50
	3,992.90	4,448.00	1,042.73	1,514.10	390.98	373.10	574.35	695.20	368.35	372.40	1,616.28	1,489.20
	4,757.52	4,967.30	1,113.97	1,536.10	586.55	457.40	618.36	708.90	461.80	456.70	1,976.49	1,804.20
	3,449.56	4,117.50	778.94	1,334.10	349.39	364.50	497.77	613.10	360.68	363.30	1,462.23	1,438.50
Avg.	**4,322.47**	**4,783.10**	**1,043.34**	**1,565.18**	**469.77**	**411.22**	**620.23**	**731.70**	**410.37**	**410.34**	**1,778.37**	**1,660.66**

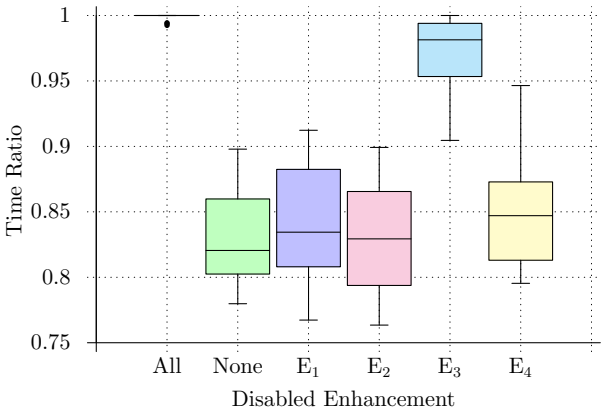


Figure 14 Time ratio boxplot for enhancement techniques

Observe that, on average, the algorithm performs faster when all enhancements are enabled. Also, E_3 is the most effective enhancement for improving the performance of the algorithm. Moreover, disabling all enhancements almost always result in the worst performance of the proposed algorithm. Finally, it seems that enhancements E_1 and E_2 are not contributing much in the performance of the proposed algorithm. To better understand the latter, we next compare '#MILPs ratios'. Let #MILPs ratio of A (for a particular instance) be the ratio of the #MILPs solved by A (for that instance) to the maximum #MILPs solved by the new algorithm (for that instance) during the entire set of experiments.

Figure 15 compares #MILPs ratios (for all 250 instances) when a particular enhancement is disabled. Observe that boxes corresponding to None, and E_1 and E_2 are almost identical ('None' is slightly better). That is mainly because that (for our test instances) the algorithm rarely faces a situation that it has to return a corner point of a triangle while splitting it.

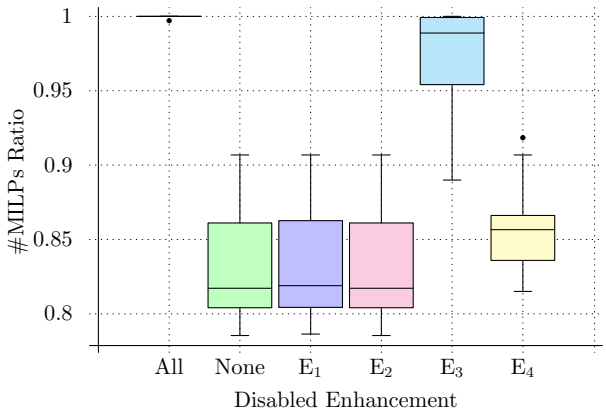


Figure 15 #MILPs ratio boxplot for enhancement techniques

Overall, from Figures 14 and 15, we observe that when all enhancements are enabled, both 'Time' and '#MILP' are improved more than 15% compared to the setting that all enhancements are disabled.

7.3. Approximate solutions

We start this section, by showing how the relative optimality gap ratio is changing for an instance of C320 during the course of the algorithm. In Figure 16, the horizontal axis shows the iterations of Algorithm 1 and the vertical axis shows the relative optimality gap ratio for an instance of C320. Note that the vertical axis shows ratios, and so, for example, '2' means '200%' (in terms of percentage). We observe that the optimality gap is decreasing reasonably fast during the course of the algorithm. More importantly, the algorithm has found an optimal solution when the gap is approximately 50%. So, basically around half of the iterations of the algorithm are just for proving that the obtained solution is actually optimal.

This particular example indicates that the proposed algorithm probably performs well if the goal is to just generate an approximate solution rather than an optimal solution. In the remaining of this section, we present the results of a series of experiments to indicate that this is actually the case. Note that since our focus is on approximations, we only use the largest class of instances, i.e., C320, in this section.

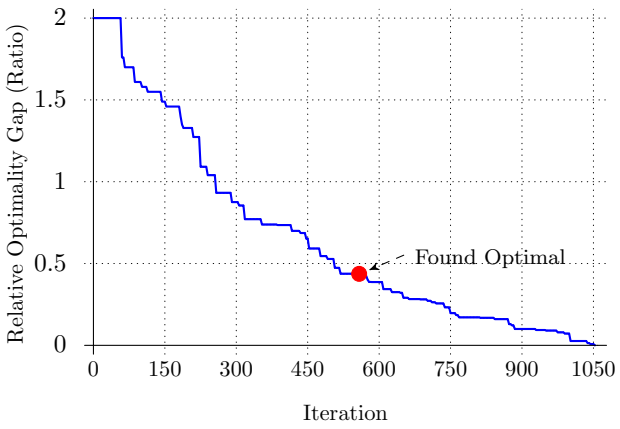


Figure 16 **Relative optimality gap obtained in each iteration**

7.3.1. Increasing the optimality gap tolerance

One way to generate an approximate solution is to increase the optimality gap tolerance, i.e., ε_2. The default value for ε_2 is 10^{-5} (which is equivalent to 0.001%). We now test the performance of the algorithm when $\varepsilon_2 \in \{10^{-3}, 10^{-2}, 10^{-1}, 5^{-1}\}$. Note that 5^{-1} is equivalent to 20%. Figure 17 shows time ratios (see the definition in Section 7.2) for different values of ε_2.

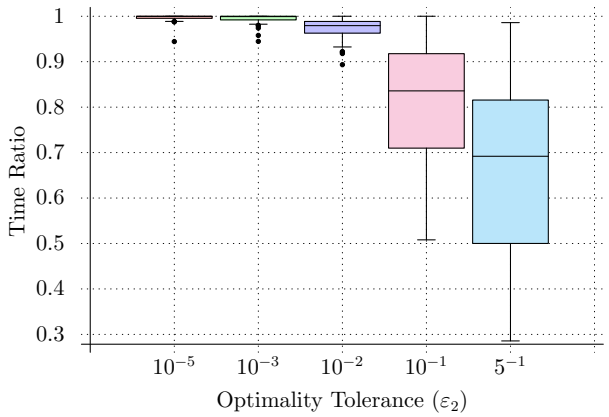


Figure 17 **Time ratio for different optimality tolerances**

Observe that there is not much difference between the boxes corresponding to $\varepsilon_2 = 10^{-5}$ and $\varepsilon_2 = 10^{-3}$. In fact, we see that for very few cases, the solution time of $\varepsilon_2 = 10^{-5}$ is actually smaller than $\varepsilon_2 = 10^{-3}$, and that is natural since the solution time can fluctuate (when we are running an experiment on a server). Overall, we observe that the solution time improves by around 2% on average by setting $\varepsilon_2 = 10^{-2}$. The improvement percentage in the solution time is around 15% and 30% on average for $\varepsilon_2 = 10^{-1}$ and $\varepsilon_2 = 5^{-1}$.

Let $(\tilde{\boldsymbol{x}}_I, \tilde{\boldsymbol{x}}_C)$ be an approximate solution and $(\boldsymbol{x}_I^*, \boldsymbol{x}_C^*)$ be an optimal solution. Note that we treat the solution generated when $\varepsilon_2 = 10^{-5}$ as an optimal solution. We define 'real optimality ratio' as $\frac{f(\boldsymbol{x}_I^*, \boldsymbol{x}_C^*)}{f(\tilde{\boldsymbol{x}}_I, \tilde{\boldsymbol{x}}_C)}$. The real optimality ratio of the generated approximate solutions for different values of ε_2 can be found in Figure 18. Observe that the quality of the approximate solution is very close to an optimal solution for all cases. In particular, we observe that even when $\varepsilon_2 = 5^{-1}$, the generated optimal solution is always almost optimal since the real optimality ratio is over 0.995. In fact, there are only very few instances that the approximate solution is up to 3% worse than an optimal solution.

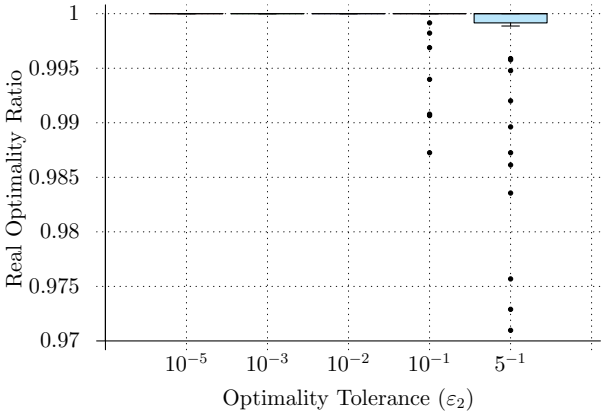


Figure 18 Real optimality ratio for different optimality tolerances

Overall, we observe from Figures 17 and 18 that we can reduce the solution time by 30% and still obtaining an almost optimal solution in practice when $\varepsilon_2 = 5^{-1}$.

7.3.2. Imposing time limit

Another way of obtaining an approximate solution is to impose a time limit for the proposed algorithm. Let t^* be the time required to compute an optimal solution (i.e., when $\varepsilon_2 = 10^{-5}$) for a particular instance. In this section, we study the consequence of imposing a time limit of αt^* where $\alpha \in \{0.05, 0.1, 0.2, 0.4, 0.8\}$ for each instance.

The relative optimality gap ratio that the algorithm is able to reach for different time limits can be found in Figure 19. Observe that by just spending around 40% of the total time limit, the relative optimality gap reaches to around 50% on average. Observe too that by spending around 80% of the total time limit, the relative optimality gap reaches to around 10% on average.

The real optimality ratio for different time limits can be found in Figure 20. Observe that even with the time limit of 5% of the total time, the quality of generated approximate solutions

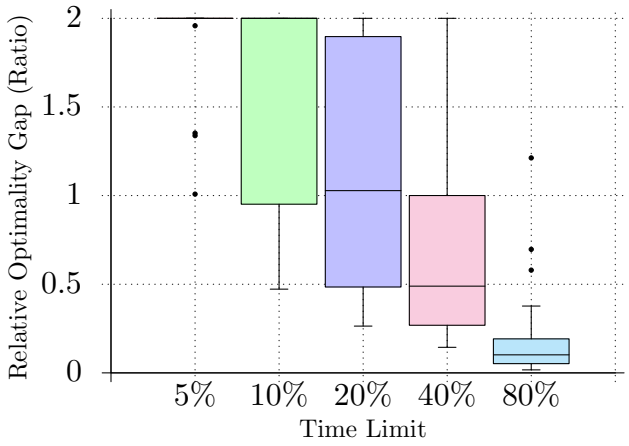


Figure 19 **Relative Optimality Gap (Ratio) for different time limits**

is only around 20% worse than the quality of optimal solutions on average. Observe too that the quality of approximate solutions quickly improves as the time limit increase. In fact, with the time limit of 20% of the total time, the quality of generated approximate solutions is almost equal to the quality of optimal solutions on average.

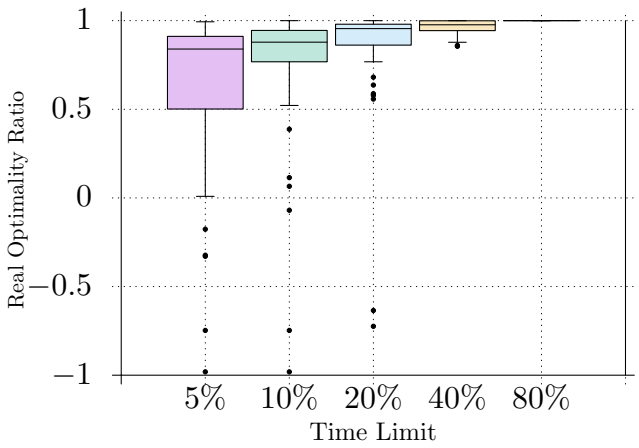


Figure 20 **Real optimality ratio for different time limits**

Overall, we observe from Figures 19 and 20 that by imposing a time limit of 20% of the total time (required to compute an optimal solution), the proposed algorithm reaches to the relative optimality gap of 100%. However, the generated approximate solutions are almost optimal in practice.

8. Conclusions

We presented the first criterion space search algorithm for optimizing a linear function over the set of efficient solutions of BOMILPs. The algorithm is based on TSM and it is easy to implement. We showed that the algorithm is fast and may explore only a small fraction

of the nondominated frontier of a BOMILP to compute an optimal solution in practice. We numerically showed that the algorithm converges quickly to an optimal solution and so it is naturally good for computing high-quality approximate solutions. Improving the proposed algorithm and developing effective parallelization frameworks for the proposed algorithm are two future research directions for this study. Moreover, there are several other exact algorithms for solving BOMILPs that can be possibly extended for optimizing a linear function over the set of efficient solutions of BOMILPs. So, exploring similar ideas for those algorithms is also another future research direction.

References

Abbas M, Chaabane D (2006) Optimizing a linear function over an integer efficient set. *European Journal of Operational Research* 174(2):1140–1161.

Aneja YP, Nair KPK (1979) Bicriteria transportation problem. *Management Science* 27:73–78.

Belotti P, Soylu B, Wiecek M (2013) A branch-and-bound algorithm for biobjective mixed-integer programs `http://www.optimization-online.org/DB_FILE/2013/01/3719.pdf`. Last accessed: September 23, 2017.

Benson HP (1984) Optimization over the efficient set. *Journal of Mathematical Analysis and Applications* 98(2):562–580.

Benson HP (1991) An all-linear programming relaxation algorithm for optimizing over the efficient set. *Journal of Global Optimization* 1(1):83–104.

Benson HP (1992) A finite, nonadjacent extreme-point search algorithm for optimization over the efficient set. *Journal of Optimization Theory and Applications* 73(1):47–64.

Benson HP (1993) A bisection-extreme point search algorithm for optimizing over the efficient set in the linear dependence case. *Journal of Global Optimization* 3(1):95–111.

Boland N, Charkhgard H, Savelsbergh M (2015a) A criterion space search algorithm for biobjective integer programming: The balanced box method. *INFORMS Journal on Computing* 27(4):735–754.

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

Boland N, Charkhgard H, Savelsbergh M (2016a) The L-shape search method for triobjective integer programming. *Mathematical Programming Computation* 8(2):217–251.

Boland N, Charkhgard H, Savelsbergh M (2016b) A new method for optimizing a linear function over the efficient set of a multiobjective integer program. *European Journal of Operational Research* Available online.

Boland N, Charkhgard H, Savelsbergh M (2016c) The quadrant shrinking method: A simple and efficient algorithm for solving tri-objective integer programs. *European Journal of Operational Research* Available online.

Chaabane D, Brahmi B, Ramdani Z (2012) The augmented weighted tchebychev norm for optimizing a linear function over an integer efficient set of a multicriteria linear program. *International Transactions in Operational Research* 19(4):531–545.

Charkhgard H, Savelsbergh M, Talebian M (2018) A linear programming based algorithm to solve a class of optimization problems with a multi-linear objective function and affine constraints. *Computers & Operations Research* 89:17 – 30.

Dächert K, Gorski J, Klamroth K (2012) An augmented weighted Tchebycheff method with adaptively chosen parameters for discrete bicriteria optimization problems. *Computers & Operations Research* 39:2929–2943.

Dächert K, Klamroth K (2014) A linear bound on the number of scalarizations needed to solve discrete tricriteria optimization problems. *Journal of Global Optimization* 1–34.

Dauer JP (1991) Optimization over the efficient set using an active constraint approach. *Mathematical Methods of Operations Research* 35(3):185–195.

Djamal C, Marc P (2010) A method for optimizing over the integer efficient set. *Journal of industrial and management optimization* 6(4):811.

Ecker J, Song J (1994) Optimizing a linear function over an efficient set. *Journal of Optimization Theory and Applications* 83(3):541–563.

Ehrgott M (2005) *Multicriteria optimization* (New York: Springer), second edition.

Fattahi A, Turkay M (2018) A one direction search method to find the exact nondominated frontier of biobjective mixed-binary linear programming problems. *European Journal of Operational Research* 266(2):415 – 425.

Gao Y, Xu C, Yang Y (2006) An outcome-space finite algorithm for solving linear multiplicative programming. *Applied Mathematics and Computation* 179(2):494 – 505.

Jorge JM (2009) An algorithm for optimizing a linear function over an integer efficient set. *European Journal of Operational Research* 195(1):98–103.

Kirlik G, Sayın S (2014) A new algorithm for generating all nondominated solutions of multiobjective discrete optimization problems. *European Journal of Operational Research* 232(3):479 – 488.

Kirlik G, Sayın S (2015) Computing the nadir point for multiobjective discrete optimization problems. *Journal of Global Optimization* 62(1):79–99.

Köksalan M, Lokman B (2015) Finding nadir points in multi-objective integer programs. *Journal of Global Optimization* 62(1):55–77.

Lokman B, Köksalan M (2013) Finding all nondominated points of multi-objective integer programs. *Journal of Global Optimization* 57(2):347–365.

Nicholson E, Possingham HP (2006) Objectives for multiple-species conservation planning. *Conservation Biology* 20(3):871–881.

Özlen M, Burton BA, MacRae CAG (2013) Multi-objective integer programming: An improved recursive algorithm. *Journal of Optimization Theory and Applications* 10.1007/s10957-013-0364-y.

Özpeynirci Ö (2017) On nadir points of multiobjective integer programming problems. *Journal of Global Optimization* Available online.

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 (2017) Multi-objective branch and bound. *European Journal of Operational Research* 260(3):856 – 872.

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

Sayin S (2000) Optimizing over the efficient set using a top-down search of faces. *Operations Research* 48(1):65–72.

Shao L, Ehrgott M (2016) Primal and dual multi-objective linear programming algorithms for linear multiplicative programmes. *Optimization* 65(2):415–431.

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

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 biobjective case. *Computers & Operations Research* 40(1):498–509.

Yamamoto Y (2002) Optimization over the efficient set: overview. *Journal of Global Optimization* 22(1-4):285.