A Criterion Space Method for Biobjective Mixed Integer Programming: the Boxed Line Method

Tyler Perini[1], Natashia Boland[1], Diego Pecin[1], and Martin Savelsbergh[1]

[1]H. Milton Stewart School of Industrial and Systems Engineering,
Georgia Institute of Technology, Atlanta

December 8, 2017

Abstract

Despite recent interest in multiobjective integer programming, few algorithms exist for solving biobjective mixed integer programs. We present such an algorithm: the Boxed Line Method. For one of its variants, we prove that the number of single-objective integer programs solved is bounded by a linear function of the number of nondominated line segments in the nondominated frontier; this is the first such complexity result. An extensive computational study demonstrates that the Box Line Method is also efficient in practice, and that it outperforms existing algorithms on a diverse set of instances.

Keywords: multiobjective, integer programming, criterion space search

1 Introduction

Biobjective optimization problems with discrete decision variables arise in many fields, including scheduling (Lei 2009), geographic information systems (Malczewski 2006), facility location (Farahani et al. 2010), health care (Rais and Viana 2011), and many more (White 1990). In contrast to single objective optimization, the goal in biobjective (and, more generally, multiobjective) optimization is to generate a *set* of solutions that induces the *nondominated frontier (NDF)*, also known as the *Pareto front*. The NDF is the set of *nondominated points (NDPs)*: an NDP is a vector of objective values evaluated at a feasible solution with the property that there exists no other feasible solution that is at least as good in all objective values and is better in one or more of them. There has been enormous interest in these problems from the evolutionary algorithms community; see, for example, the surveys of Coello (2000), Deb (2001), Zhou et al. (2011). In this paper, we focus on problems with *linear* objective functions and constraints, and on *exact* algorithms, which are guaranteed, in theory, to produce the complete NDF.

Biobjective integer linear programming has been studied for several decades now; see, for example, the surveys of Ehrgott and Gandibleux (2000), Gandibleux (2006). However, to date much more attention has been paid to the development of computationally effective algorithms for *pure* integer linear programming problems than to problems that mix continuous and discrete variables. Over the last thirty years, at least a dozen methods for generating the NDF for pure integer problems have been developed.

Biobjective mixed integer linear programs (BOMIPs), on the other hand, have only recently received vigorous interest, in part due to their additional numerical challenges. BOMIP frontiers have a complex structure, with continuous segments and discontinuities (vertical and horizontal gaps). The frontier can contain closed, open, and half-open line segments, as well as isolated points; see, for example, the NDF illustrated in Figure 1. Numerical tolerances, and how these are used within an algorithm, can significantly affect the representation of the NDF it produces.

Algorithms based on branch-and-bound, working in the space of the decision variables, are given by Mavrotas and Diakoulaki (1998), Mavrotas and Diakoulaki (2005) and Vincent et al. (2013). Combining branching on the decision variables with Pareto branches, which work in the criterion space, Stidsen et al. (2014) develop a method designed for the special case that only one of the two objective functions has continuous variables. In this case, the NDF has a special structure: it consists of isolated points; no line segments can occur.

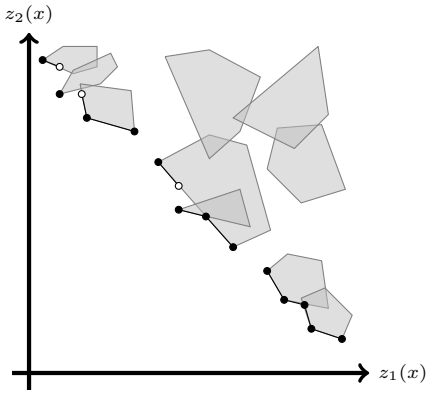


Figure 1: The nondominated frontier of a BOMIP where nondominated line segments are darkened.

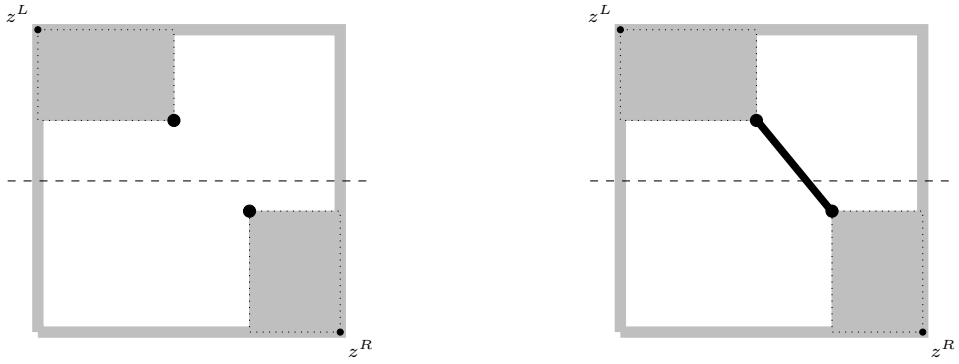
Recently, criterion space methods, in which the search for the NDF operates over the space of the vector of objective function values, known as the *criterion space*, have emerged. Such methods have the advantage of being able to exploit advances in single-objective solver software, since these methods repeatedly solve single-objective problems, both linear programs (LPs) and mixed integer linear programs (which we will generically refer to as *IPs*), treating the single-objective solver as a "black box". Single-objective problems, either LP or IP, are the main "workhorses" of these algorithms, which differ mainly in the structure and number of such problems that need to be solved before the NDF is completely identified.

The first of these algorithms to be published is the *Triangle Splitting Algorithm (TSA)* (Boland et al. 2015b). The TSA first identifies all extreme supported NDPs, using dichotomic search (Aneja and Nair 1979), which allows the remaining search region to be divided into right-angled triangles. Each triangle is then split, either horizontally or vertically, with each half searched for an NDP so as to reduce the remaining search region within the triangle to two rectangles. Within each rectangle, all extreme supported NDPs are found (these are extreme and supported only in the local sense, within the rectangle), and the process repeats. Line segments in the NDF are identified when they form part of the hypotenuse of a triangle. TSA may split a line segment in the NDF. This can occur even if the line segment is part of the frontier of a unique *slice problem*, which is a BOLP defined by fixing the integer part of the solution. The TSA thus requires a post-processing procedure to provide a parsimonious description of the NDF.

The second algorithm published is the ϵ*, Tabu-Constraint Method (*ϵ*TCM)* of Soylu and Yıldız (2016), which uses "tabu", or "no-good" constraints, to identify line segments in the NDF, combined with ϵ-constraints to progressively generate the frontier from right-to-left (or vice versa).

In this paper, we make the following key contributions.

1. We propose a new criterion space search method for solving the BOMIP: the Boxed Line Method. The method is designed to generalize the *Balanced Box Method (BBM)* (Boland et al. 2015a), which is a computationally effective method for pure integer BOIP. The Boxed Line Method defaults to BBM in the absence of continuous variables. The key step of the BBM is illustrated in Figure 2a: the rectangular search region (box) is split, each half is searched for an NDP, which reduces the remaining search region to two boxes having total area less than half that of the original box. To apply this idea to BOMIP, we observe that when the split line passes through a line segment of the frontier, the NDP found when the first half of the box is searched will lie on the split line. In the Boxed Line Method, we seek to extend this NDP to the line segment in the NDF which contains it. Using the end points of this line segment again gives a remaining search region consisting of two boxes, with combined area less than half that of the original box. Figure 2b illustrates this idea. As a consequence of the idea, the resulting algorithm is amenable to analysis (discussed next) and produces a parsimonious description of the NDF.
2. The algorithm has two variants that permit analysis of the number of single-objective IPs that they require to be solved: a basic, iterative version and a recursive version. For both variants, we provide upper bounds on the number of single-objective IPs required to produce the NDF. These are the first analytic results of this type for mixed integer multiobjective problems.
3. We design enhanced variants of the basic iterative version, which better engineer the algorithm for



(a) Nondominated points are filled.

(b) Nondominated line segments are darkened.

Figure 2: The key step in the Balanced Box Method and the Boxed Line Method, respectively. The remaining search regions (boxes) after the step are shaded

practical computation, and are more amenable to parallel implementation.

4. The benchmark instances originally proposed by Mavrotas and Diakoulaki (1998), which have been used to test recent methods, have a structure that is very sensitive to the numerical tolerances used in algorithms. In these instances, the slice problems often have many short line segments in their frontiers, and include line segments that are close to horizontal or close to vertical. The instances also have frontiers represented by a relatively small number of slice problems, which can bias comparisons of algorithms. Here, we propose a new approach to creating instances, in a controlled way. The approach facilitates validation of BOMIP algorithms, by producing instances for which the frontiers are known precisely, *a priori*. It also supplements the existing suite of test instances by providing instances having different characteristics, such as having many slice problems contributing to the frontier.
5. We provide computational results that demonstrate the relative strengths and weaknesses of the Boxed Line Method variants on two classes of the new instances proposed here, as well as on instances from Mavrotas and Diakoulaki (1998). The results in the latter case are compared with the results of the TSA (Boland et al. 2015b) and in both cases with the ϵTCM (Soylu and Yıldız 2016).

The paper is structured as follows. In Section 2, we introduce notation and key definitions. The basic variant of the Boxed Line Method is described in Section 3. In Section 3.5, we prove that the method has an upper bound for the number of IPs. We continue by presenting the recursive extension to the method (Section 4) along with its better upper bound for the number of IPs (Section 4.4). The enhanced version of the basic method, engineered for practical implementation, is discussed in Section 5. In Section 6, we discuss our observations on the structure of existing benchmark instances, and their implications. Then in Section 7, we present our new instance generator method. Finally, we give our computational study in Section 8 and summarize our findings.

2 Definitions

We define the *biobjective mixed integer linear program (BOMIP)* as

$$\min_{x \in \mathbb{X}} \{z(x) := (z_1(x), z_2(x))\} \tag{1}$$

where $z_1(x), z_2(x)$ are linear in x and the feasible region is given by $\mathbb{X} \subseteq \mathbb{Z}^n \times \mathbb{R}^m$. We assume $\mathbb{X}$ is nonempty and bounded. Note that a vector $x \in \mathbb{X}$ has n integer components and m continuous components, and when it is convenient to differentiate between the components, we use the convention $x = (x_I, x_C)$ where $x_I \in \mathbb{Z}^n$ and $x_C \in \mathbb{R}^m$. Let the projections of $\mathbb{X}$ onto the set of integers and reals be defined as $\mathbb{X}_I := \{x_I \in \mathbb{Z}^n : (x_I, x_C) \in \mathbb{X} \text{ for some } x_C \in \mathbb{R}^m\}$ and $\mathbb{X}_C := \{x_C \in \mathbb{R}^m : (x_I, x_C) \in \mathbb{X} \text{ for some } x_I \in \mathbb{Z}^n\}$, respectively. The feasible region $\mathbb{X}$ lies in the *decision space*, $\mathbb{Z}^n \times \mathbb{R}^m$, or simply $\mathbb{R}^{n+m}$.

On the other hand, the image of $\mathbb{X}$ under $z(x)$, i.e. $\mathcal{Y} := \{y \in \mathbb{R}^2 : y = z(x) \text{ for some } x \in \mathbb{X}\}$, lies in the *criteria space*, $\mathbb{R}^2$.

Let $x^1, x^2 \in \mathbb{X}$ be two feasible solutions to (1). If $z_i(x^1) \leq z_i(x^2)$ for $i = 1, 2$ and $z(x^1) \neq z(x^2)$, then we say that $z(x^1)$ *dominates* $z(x^2)$. If $x^N \in \mathbb{X}$ and there does not exist feasible solution $x \in \mathbb{X}$ such that $z(x)$ dominates $z(x^N)$, then $z(x^N)$ is a *nondominated point* (NDP) and feasible solution x^N is *efficient*. The union of all nondominated points is defined as the *nondominated frontier* (NDF), which we represent with $\mathcal{N}$.

A single NDP of (1) can be found by solving single-objective IPs over $\mathbb{X}$ either by lexicographic optimization or by use of a scalarized objective function. We use the term *integer program (IP)* to refer to any single-objective problem that has integer variables, including mixed integer linear programs. The *lexicographic IP* hierarchically optimizes over two objectives in turn. In the case of minimizing $z_1(x)$ and then $z_2(x)$, we denote it by

$$\eta = \text{lexmin}\{(z_1(x), z_2(x)) : \ x \in \mathbb{X}\}. \tag{2}$$

Solving a lexicographic IP computationally requires solving 2 IPs in sequence. In this case,

$$\eta_1 = \min\{z_1(x) : \ x \in \mathbb{X}\} \ \text{ and then } \ \eta_2 = \min\{z_2(x) : \ z_1(x) \leq \eta_1, x \in \mathbb{X}\}$$

are solved. Then $\eta = (\eta_1, \eta_2)$ is an NDP of (1). We note that in practice this second IP tends to solve very quickly. Lexicographic optimization in the order $z_2(x)$ then $z_1(x)$ is similar. For vector $\lambda \in \mathbb{R}^2_+$, $\lambda_1, \lambda_2 > 0$, the *scalarized IP with respect to* λ is defined as

$$\min\{\lambda^T z(x) : \ x \in \mathbb{X}\}. \tag{3}$$

Let x^* be an optimal solution to (3). Then $z(x^*)$ is an NDP of (1). Not every NDP in the frontier can be found by such scalarization: if, for a given nondominated point $z(x_N)$, there exists positive vector $\lambda \in \mathbb{R}^2_+$ such that x^N is an optimal solution to (3), then the NDP is *supported*; otherwise, the NDP is *unsupported*.

The nondominated frontier can be described by nondominated line segments, vertical gaps, and horizontal gaps. Define $L(z^1, z^2)$ to be the line segment connecting endpoints $z^1, z^2 \in \mathbb{R}^2$, i.e. $L(z^1, z^2) := \{\lambda z^1 + (1 - \lambda)z^2 : 0 \leq \lambda \leq 1\}$ where the endpoints are ordered so that $z^1_1 \leq z^2_1$, and each endpoint is flagged as open or closed. When z^1 is flagged as open, the upper bound on λ is excluded (i.e., $\lambda < 1$), and when z^2 is flagged as open, the lower bound on λ is excluded (i.e., $0 < \lambda$). A line segment $L(z^1, z^2)$ is a point if $z^1 = z^2$.

We define S to be the index set of all feasible integer solutions in $\mathbb{X}_I$. In other words, S is the index set of all slice problems. The LP nondominated frontier of a slice problem consists of a (connected) set of (closed) line segments; we call this a *slice*. For slice problem with index $s \in S$, we denote its NDF by N^s (this is a slice). If a slice problem with index $s \in S$ contributes to the BOMIP NDF, so $N^s \cap \mathcal{N}$ is nonempty, then we write $N^s \cap \mathcal{N} = \{L^s_1, L^s_2, ..., L^s_{n(s)}\}$, where $n(s)$ is the number of line segments contributed to the nondominated frontier by the slice problem. (If $N^s \cap \mathcal{N}$ is empty, then $n(s) = 0$.) Each of the line segments, L^s_m for some $s \in S$ and $m = 1, \ldots, n(s)$, is a *nondominated line segment* (NLS) in the NDF. Note that a single (maximal) line segment in N^s may contribute several NLSs to the NDF, consisting of non-overlapping sections of the line segment. We make the natural assumption that the set $N^s \cap \mathcal{N}$ consists of maximal line segments, so $N^s \cap \mathcal{N}$ provides the minimum cardinality set of line segments that describes the slice's intersection with the BOMIP frontier. We now define the total number of NLSs in a given BOMIP frontier as

$$n := \sum_{s \in S} |N^s \cap \mathcal{N}|. \tag{4}$$

A gap in both vertical and horizontal directions may appear in the frontier between two NDPs. Between an NDP and an open endpoint of an NLS there must appear either a vertical or a horizontal gap. We define a *vertical gap* as an interval $(y^-, y^+) \subset \mathbb{R}$ such that no nondominated point p exists with $p_2 \in (y^-, y^+)$ but where there does exist an NDP p^- with $p^-_2 = y^-$ and either an NDP p^+ with $p^+_2 = y^+$ or a sequence of NDPs $\{p^0, p^1, ...\}$ with $\lim_{n\to\infty} p^n_2 = y^+$ (or both). A *horizontal gap* can be defined similarly.

3 Basic Method

Here we present the fundamental principles of the Boxed Line Method. The four main components of the algorithm are initialization, the outer loop, the inner loop, and the line segment generation subroutine.

For this presentation, we assume exact arithmetic. For example, we describe some constraints as strict inequalities. These are implemented, in practice, as inequalities with the right-hand side adjusted using the desired accuracy, ϵ. The pseudocode provided in the appendix makes this explicit. In practice, the numerical issues naturally encountered in solving integer programs, which are even more pressing for mixed integer programs, make it unreasonable to expect to determine the frontier exactly. In Section 6, we define an approximation – the ϵ-nondominated frontier – and discuss the numerical challenges that arise in finding it.

3.1 Initialization

The first stage, initialization, solves two lexicographic IPs to determine the upper-leftmost NDP, $z^L = \text{lexmin}\{(z_1(x), z_2(x)) : x \in \mathbb{X}\}$, and the lower-rightmost NDP, $z^R = \text{lexmin}\{(z_2(x), z_1(x)) : x \in \mathbb{X}\}$. The entire nondominated frontier must lie in the box with z^L and z^R as its two corner points, which we denote by $B(z^L, z^R)$. Note that if $z^L = z^R$, then $B(z^L, z^R)$ is a point. We call such a box *trivially small*. If the box $B(z^L, z^R)$ is trivially small, then the method terminates and the point is returned as the full nondominated frontier. Otherwise, the two points are added to the current subset of the nondominated frontier, which we call $\mathcal{N}$, and the box $B(z^L, z^R)$ is added to the queue, which we call $\mathcal{Q}$, for further processing by the outer loop. In general, the basic method only adds boxes to the queue if both corner points are the closed endpoints of a NLS to which they belong (we discuss this further in Section 3.5).

3.2 Outer Loop

To introduce the outer loop of the basic version of the Boxed Line Method, we first describe the process for the Balanced Box Method (Boland et al. 2015a) because, in the first case we consider, the two algorithms follow the same procedure. The steps are visually summarized in Figure 3.

The outer loop is defined as a while loop that ends once $\mathcal{Q}$ is empty. The main roles of the outer loop are to remove boxes from $\mathcal{Q}$ for processing, split the boxes to begin processing, (if necessary) call the inner loop to complete processing, and update $\mathcal{N}$ with any found NDPs and $\mathcal{Q}$ with remaining boxes for future processing. Suppose a (nontrivial) box $B(z^L, z^R)$ is arbitrarily chosen from $\mathcal{Q}$ where z^L and z^R now represent the corner points of the new box, not not necessarily the NDPs found by initialization. The outer loop begins processing by choosing an arbitrary horizontal split line $\mu \in (z_2^L, z_2^R)$. We solve a lexicographic IP for an NDP on or below the split line, namely

$$z^* = \text{lexmin}\{(z_1(x), z_2(x)) : z_2(x) \leq \mu, x \in \mathbb{X}\}. \tag{5}$$

We note that the solution $x^R \in \mathbb{X}$ which maps to $z^R = z(x^R)$ is feasible for (5), so in practice we provide x^R as an initial feasible solution to the solver. Since the frontier of a pure IP is a union of discrete points, the NDP z^* will likely be strictly below the split line, i.e. we expect $z_2^* < \mu$ (it is unlikely that the split line intersects an NDP exactly). The next step is to form a second split line, this time vertically at z_1^* and solve a second lexicographic minimization for the next NDP,

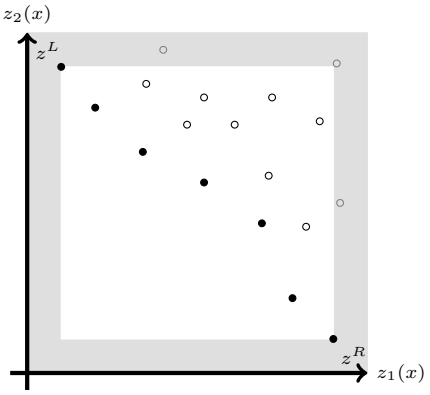
$$\hat{z} = \text{lexmin}\{(z_2(x), z_1(x)) : z_1(x) < z_1^*, x \in \mathbb{X}\}. \tag{6}$$

The strict inequality used here is a convenient shorthand: it is meant to be interpreted and implemented as $z_1(x) \leq z_1^* - \epsilon$ for some $\epsilon > 0$. The solution $x^L \in \mathbb{X}$ that maps to $z^L = z(x^L)$ is feasible for (6), so in practice, x^L is provided as an initial feasible solution to the solver.

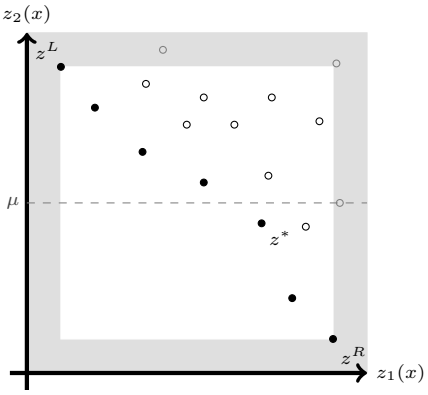
Finally, the outer loop updates $\mathcal{N}$ and $\mathcal{Q}$, accordingly. First, z^* and $\hat{z}$ are added to $\mathcal{N}$. Then, z^* and $\hat{z}$ are used respectively with z^R and z^L to define two new boxes, $B(z^L, \hat{z})$ and $B(z^*, z^R)$, each of which is added to the queue if it is not trivially small. This concludes the processing of one box by the outer loop when $z_2^* < \mu$.

The Boxed Line Method follows the above procedure when the first lexicographic minimization (5) returns NDP z^* such that $z_2^* < \mu$. Note that when there are continuous variables, a discontinuous nondominated frontier implies that in some cases it is still possible for an arbitrary horizontal split line μ to not intersect any NDP, which results in $z_2^* < \mu$. In this case, we say that the outer loop has *identified a vertical gap* in the frontier. Once a vertical gap is identified by the outer loop, the boxes resulting from processing are such that this vertical gap is not in either of the boxes added to the queue (we prove this formally in Section 3.5).

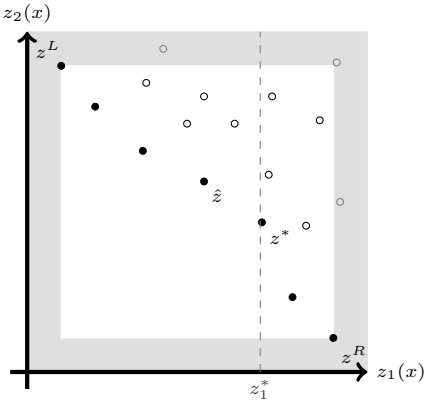
In a *mixed* IP with continuous variables it is likely that the horizontal split line will intersect an NLS of the nondominated frontier. In this case, the first lexicographic minimization (5) returns NDP z^* such that $z_2^* = \mu$. The rest of the procedure is designed to *identify the NLS* that contains z^*, (as it intersects



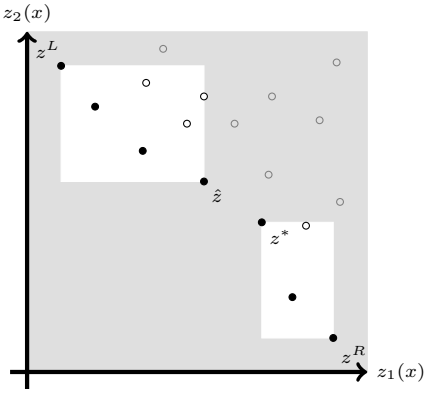
(a) Initialization identifies z^L and z^R and defines the white box to the queue. The outer region in grey is ignored.



(b) A horizontal split line μ is arbitrarily chosen. Lexicographically minimizing $z_1(x)$ then $z_2(x)$ below the split line yields z^* where $z_2^* < \mu$.



(c) A vertical split line is chosen at z_1^*. Lexicographically minimizing $z_2(x)$ then $z_1(x)$ (strictly) to the left of the split line yields $\hat{z}$.



(d) The frontier approximation, $\mathcal{N}$, is updated with $\hat{z}$ and z^*, and the two new boxes (in white) are added to the queue, $\mathcal{Q}$.

Figure 3: Outer loop procedure for the Balanced Box Method and the Boxed Line Method when $z_2^* < \mu$.

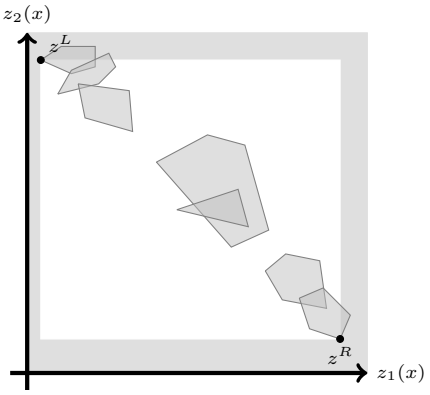
the box), i.e., to find $L(z^1, z^2) \subset B(z^L, z^R)$ such that $z^* \in L(z^1, z^2)$. This concept motivates the name for the Boxed Line Method.

The outer loop calls the inner loop, which returns $L(z^1, z^2)$ (details on how this is done are given in the next section). The inner loop returns endpoints z^1 and z^2, as well as the NDP that dominates each open endpoint, if any. The outer loop updates the current approximation of the frontier by adding $L(z^1, z^2)$ to $\mathcal{N}$ and updates the queue by adding up to two nontrivial boxes. Since our method only creates boxes in which both corner points are nondominated, we initialize the boxes depending on the openness of the endpoints as follows. If z^1 is closed, then $B(z^L, z^1)$ is added to $\mathcal{Q}$. Otherwise the inner loop has returned an NDP, $\hat{z}^1$, that dominates z^1, and the box $B(z^L, \hat{z}^1)$ is added to $\mathcal{Q}$. Similarly, if z^2 is closed, then $B(z^2, z^R)$ is added to $\mathcal{Q}$. Otherwise the inner loop has returned an NDP, $\hat{z}^2$, that dominates z^2, and the box $B(\hat{z}^2, z^R)$ is added to $\mathcal{Q}$.

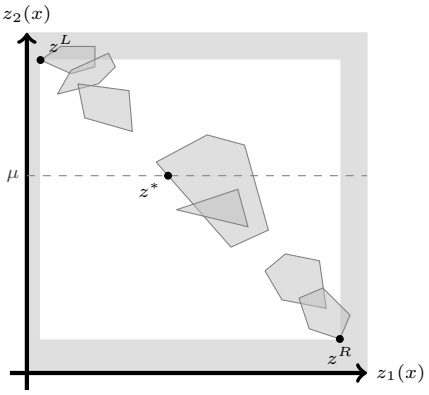
This concludes the processing of one box by the outer loop when $z_2^* = \mu$, which is summarized in Figure 4. The pseudocode is included in the Appendix, as Algorithm 1.

3.3 Inner Loop

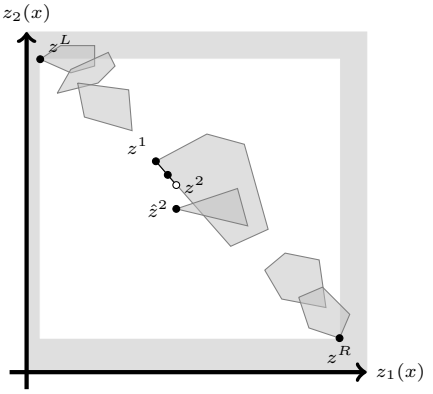
The inner loop is called when the outer loop is processing box $B(z^L, z^R)$ and the split line contains an NDP, z^*, i.e. when $z_2^* = \mu$. The inner loop finds the NLS, $L(z^1, z^2)$, that contains z^*. Its steps are visually summarized in Figure 7, and the pseudocode is presented in the Appendix, as Algorithm 2. Here



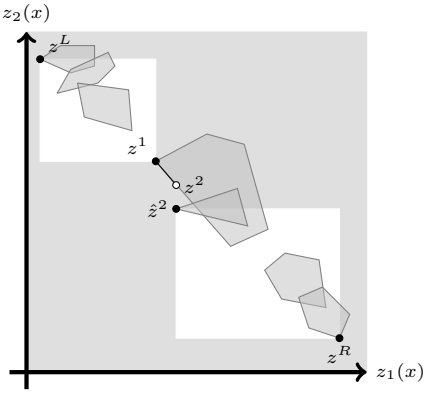
(a) Suppose the box $B(z^L, z^R)$ is chosen from $\mathcal{Q}$.



(b) Minimizing $z_1(x)$ then $z_2(x)$ below the split line μ yields z^* where $z_2^* = \mu$.



(c) The Inner Loop returns the nondominated line segment containing z^*, indicating whether each endpoint z^1 and z^2 are either open or closed, and any point that may dominate its open endpoint(s), e.g. $\hat{z}^2$ above.



(d) The frontier approximation is updated with line segment $L(z^1, z^2)$, and two new boxes are added to the queue, e.g., $B(z^L, z^1)$ and $B(\hat{z}^2, z^R)$ (assuming they are nontrivial).

Figure 4: Outer loop procedure for the Boxed Line Method when $z_2^* = \mu$.

we provide a description.

Let $x^* = (x_I^*, x_C^*) \in \mathbb{X}$ be a solution that maps to $z^* = z(x^*)$. The first step in the inner loop is to provide an *overestimation* for the NLS by generating the line segment containing z^* in the frontier of the slice problem for x_I^*, restricted to $B(z^L, z^R)$.We discuss the details of the line segment generation subroutine in the next section. Here we assume that $L(z^1, z^2)$, the maximal line segment with $z^* \in L(z^1, z^2)$, $z_1^L \le z_1^1 \le z_1^2 \le z_1^R$, $z_2^L \ge z_2^1 \ge z_2^2 \ge z_2^R$, and z^1, z^2, and z^* all NDPs of the slice problem for integer vector, x_I^*, is given.

Since $L(z^1, z^2)$ is part of the NDF of the slice problem (i.e., of the slice) for x_I^*, it must be that either (i) $z^1 = z^2$ or (ii) $z_1^1 < z_1^2$ and $z_2^1 > z_2^2$. We consider theses two cases in turn.

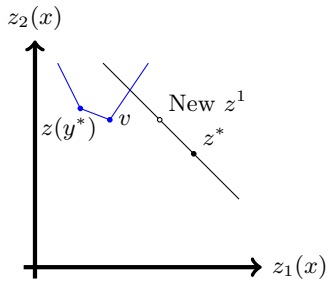
In the case that $z^1 = z^* = z^2$, the slice consists of the isolated NDP z^*, so the inner loop ends and returns $z^1 = z^*$ and $z^2 = z^*$ (both closed).

Otherwise, $z^1 \neq z^2$, and we may define the gradient vector of the line segment, $L(z^1, z^2)$, normalized to have unit length, as

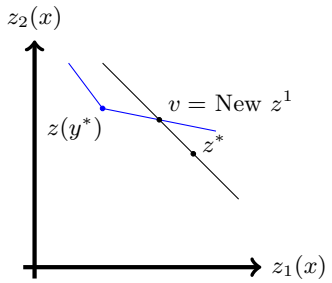
$$\vec{w} = (z_2^1 - z_2^2, z_1^2 - z_1^1)/||(z_2^1 - z_2^2, z_1^2 - z_1^1)||.$$

In what follows, the objective function $\vec{w}(z_1(x), z_2(x))$ is used for scalarized IPs.

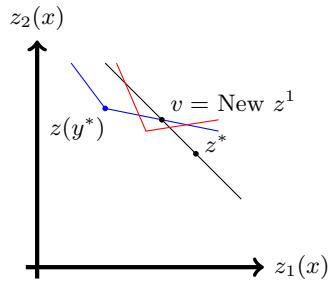
Next, the endpoints of $L(z^1, z^2)$ are restricted, iteratively, until only the nondominated portion of it remains. Either or both ends are trimmed while always maintaining z^* as the "central point" between z^1 and z^2, ensuring that the final line segment contains z^*. Until the inner loop terminates, each iteration of the loop updates the endpoints of the line segment, z^1 and z^2, (both their coordinates and their open/closed flag), with any newfound knowledge. For instance, immediately after the line generation



(a) The frontier of y_I^* is in blue. Minimizing $z_2(x)$ on this frontier finds v. Updating the upper endpoint to z^1 as shown (which is open) will prevent y_I^* recurring when solving (7).



(b) The minimization to find v along the frontier of y_I^* cannot surpass $L(z^1, z^2)$ because of the constraint to lie on or below the current line segment. Since $v \in L(z^1, z^2)$, z^1 is updated as $z^1 = v$ and flagged to be closed.



(c) The point v in this example is dominated by part of another integer solution's frontier (in red).

Figure 5: Observations about the construction of LP (8).

subroutine provides $L(z^1, z^2)$, both endpoints are flagged as closed unless the following cases occur: (i) $z_1^R = z_1^2$ and $z_2^R < z_2^2$, illustrated in Figure 7a, in which case the endpoint z^2 is dominated by z^R, so z^2 cannot be closed and is flagged as open, and (ii) $z_2^L = z_2^1$ and $z_1^L < z_1^1$, in which case z^1 is flagged as open.

The inner loop is a while loop that continues until the entirety of the current line segment, $L(z^1, z^2)$, is nondominated. To check this stopping condition, the following scalarized IP,

$$z(y^*) = \min\{\vec{w}^T z(x) : z_1(x) \leq z_1^2, \quad z_2(x) \leq z_2^1, \quad x \in \mathbb{X}\}, \tag{7}$$

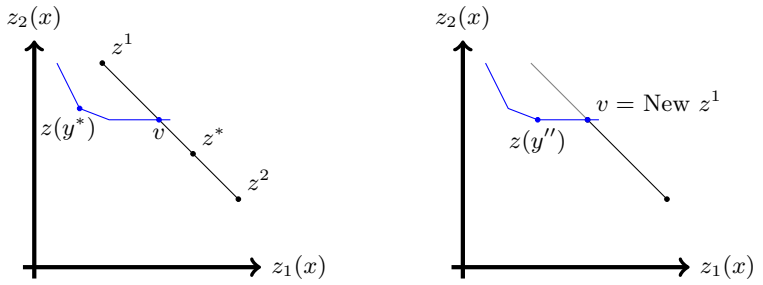
is solved, where the first inequality is strict if z^2 is open and the second inequality is strict if z^1 is open. These *conditionally strict* inequalities are required to avoid cycling. Now $L(z^1, z^2)$ is nondominated if and only if (7) yields an optimal solution, y^*, with $\vec{w}^T z(y^*) = \vec{w}^T z^*$, i.e., if and only if any NDP within the region $B(z^1, z^2)$ lies on $L(z^1, z^2)$. Thus, $\vec{w}^T z(y^*) = \vec{w}^T z^*$ is an appropriate stopping criterion for the while loop.

In the case that $\vec{w}^T z(y^*) \neq \vec{w}^T z^*$ it must be that $\vec{w}^T z(y^*) < \vec{w}^T z^*$, since z^* in $L(z^1, z^2)$ ensures that the feasible solution mapping to z^* is feasible for (7). Thus, the NDP $z(y^*)$ dominates a portion of the current line segment. There are two cases: either $z(y^*)$ is to the left or it is to the right of z^*. Without loss of generality assume that $z_1(y^*) < z_1^*$ (the other case follows from a symmetric argument). Now there may be many other points from the slice of y_I^* that also dominate part of $L(z^1, z^2)$: we update z^1 so as to exclude all such points from the box induced by the current line segment, by finding the point with integer solution y_I^* that dominates part of $L(z^1, z^2)$ and has minimal z_2-coordinate. Thus we solve the LP

$$\min\{z_2(x) : \ \vec{w}^T z(x) \leq \vec{w}^T z^*, \quad x_I = y_I^*, \quad x \in \mathbb{X}\}. \tag{8}$$

Let y' be an optimal solution to (8) and define $v = z(y')$. We make some important observations about the LP defined in (8).

- It minimizes $z_2(x)$ to ensure all points on the y_I^* frontier that can dominate the current line segment are excluded when z_2^1 is updated. The idea is illustrated in Figure 5a.
- Even though the point v may lie on the line segment $L(z^1, z^2)$, it will never be *dominated* by $L(z^1, z^2)$ because of the first constraint, $\vec{w}^T z(x) \leq \vec{w}^T z^*$. Consider Figure 5b.
- While the first scalarization returns an NDP $z(y^*)$, the point v is not necessarily nondominated. Consider Figure 5c. This issue will be discussed later.
- As it is an LP, (8) can be solved efficient. Furthermore, since the solution y^* is feasible for (8), in implementation we provide this initial feasible solution to the solver, which yields even greater efficiency.



(a) $z(y^*)$ dominates the line segment. The LP (8) has multiple optimal solutions. Finding v on the current line segment requires checking for an alternative optimum before updating the open/closed flag for z^1.

(b) Solving (9) yields y'' with $z_1(y'') < v_1$, so the updated z^1 is flagged to be open.

Figure 6: The case that v lies on the current line segment and is a weak NDP of the slice problem for y^*.

Next, v is used to update z^1. First, suppose $v \notin L(z^1, z^2)$. Point $p \in L(z^1, z^2)$ such that $p_2 = v_2$ is computed and then z^1 is updated: $z^1 = p$ and flagged to be open (see Figure 5a). Otherwise, $v \in L(z^1, z^2)$ and z^1 is updated to $z^1 = v$. Before updating the open/closed flag for z^1, we need to check for the possibility that v is a weakly nondominated point on the slice for y_I^*: if it is, then the new z^1 will be open and otherwise it will be closed. To check this, we solve a second LP,

$$\min\{z_1(x) : \ z_2(x) = v_2, \quad x_I = y_I^*, \quad x \in \mathbb{X}\}. \tag{9}$$

Let y'' be a solution of (9) and update $v = z(y'')$. (Note that even this updated v is still not necessarily an NDP of the BOMIP). If $v_1 < z_1^1$ then flag z^1 as open and otherwise flag it as closed.

After updating z^1 as described, then, by convexity, it is guaranteed that no point from the slice of y_I^* that dominates part of $L(z^1, z^2)$ can again be found. Since $\mathbb{X}$ is bounded, which implies that $\mathbb{X}_I$ is finite, the inner loop will terminate in a finite number of iterations. The resulting endpoints z^1 and z^2 will then define the NLS containing z^*, $L(z^1, z^2)$.

As a final step, the inner loop finds each NDP that dominates an endpoint that is open. Let $\hat{z}^1$ denote the NDP that dominates z^1 when z^1 is open, and let $\hat{z}^2$ denote the NDP that dominates z^2 when z^2 is open. To determine $\hat{z}^i$ for $i \in \{1, 2\}$, two special cases are checked first. (1) If $z_2^1 = z_2^L$, then clearly $\hat{z}^1 = z^L$ (and similarly for z^2 and z^R). (2) If $z_2^1 = z_2(y^*)$ then $\hat{z}^1 = z(y^*)$ (see Figure 7d) (and similarly if $z_1^2 = z_1(y^*)$ then $\hat{z}^2 = z(y^*)$). Otherwise, the inner loop identifies these NDPs by solving the IP

$$\min\{z_1(x) : z_2(x) \leq z_2^1, \ x \in \mathbb{X}\}, \tag{10}$$

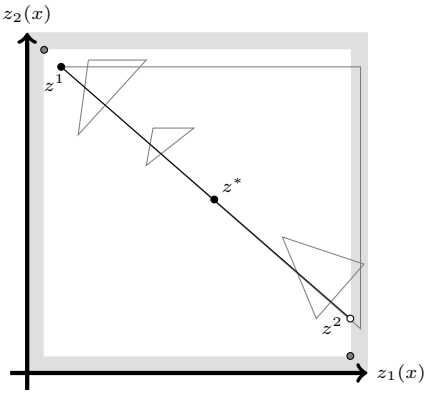
if z^1 is open, and by solving the IP

$$\min\{z_2(x) : z_1(x) \leq z_1^2, \ x \in \mathbb{X}\}, \tag{11}$$

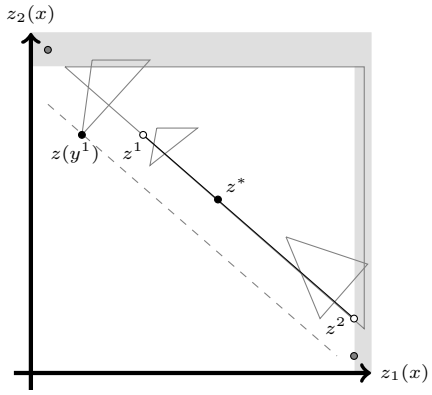
if z^2 is open. If z^1 is open, $\hat{z}^1 = z(x^1)$, where x^1 is the solution to (10), and if z^2 is open, $\hat{z}^2 = z(x^2)$, where x^2 is the solution to (11).

3.4 Line Segment Generation Subroutine

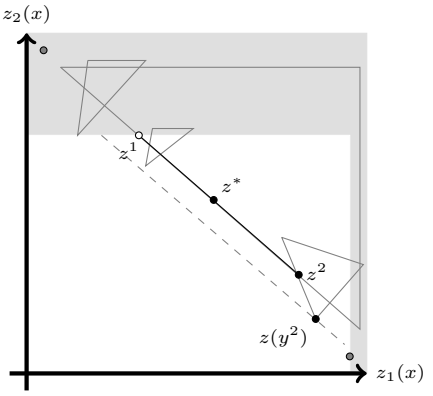
This subroutine determines an overestimate of the nondominated line segment containing a given NDP, $z^* = z(x^*)$, within $B(z^L, z^R)$. Specifically, it finds a maximal line segment, $L(z^1, z^2)$, of the slice problem for given integer solution x_I^* with $z^* \in L(z^1, z^2)$. Such a segment is an NLS of the slice problem $\min\{(z_1(x), z_2(x)) : x_I = x_I^*, x \in \mathbb{X}\}$. Figure 8 illustrates the three possible cases for z^*: z^* may be an isolated point or it may belong to one or two nondominated line segments. When z^* belongs to two nondominated line segments, the subroutine find the segment to the lower right, i.e., it takes $z^1 = z^*$ (see Figure 8c).



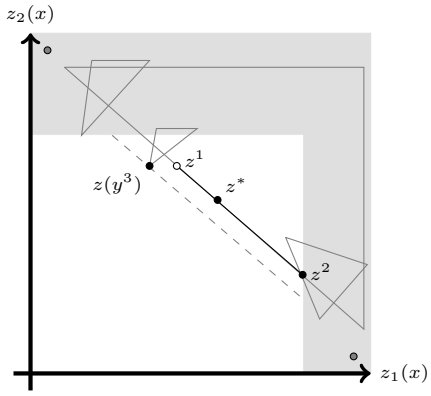
(a) The gray corner points of the white box are z^L and z^R. The darkened line is the approximation of the line segment immediately after line generation subroutine. Note that z^2 is open because it lies directly above z^R: $z_1^2 = z_1^R$ and $z_2^2 > z_2^R$.



(b) Scalarization by the slope of the line segment (within the white region bounded by the previous z^1,z^2) identifies $z(y^1)$, which is also equal to v. We update z^1 accordingly, so it is now open.



(c) Scalarization by the slope of the line segment (within the white region bounded by the previous z^1,z^2) identifies $z(y^2)$. However, v is at the intersection of the integer's frontier with the line segment, so it becomes the new z^2 which is now closed.

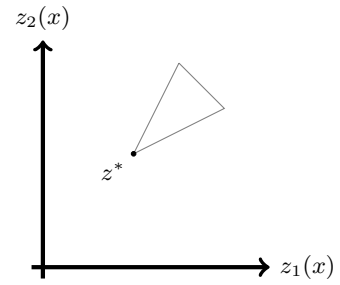


(d) Scalarization yields $z(y^3)$, then solve for $v = z(y^3)$. Update z^1, and one final scalarization confirms that $L(z^1, z^2)$ is nondominated. Although z^1 is open, in this case the NDP that dominates it is already known, i.e. $\hat{z}^1 = z(y^3)$.

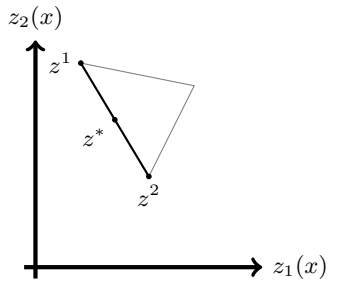
Figure 7: The inner loop takes as input NDPs z^L, z^R, and z^*. The output is the (maximal) nondominated line segment containing z^*, $L(z^1, z^2)$, and the NDP that dominates any open endpoint.

Finding such a line segment can be done, in theory, by accessing the values of the dual variables at the optimal branch-and-bound node of the IP that found z^* and applying LP duality theory. However, we are not confident that, in practice, the IP solver will provide the LP dual variables needed, since it may have found z^* with a primal heuristic, eliminated variables and/or constraints or modified coefficients with preprocessing, or added constraints. Here, we simply use a variant of dichotomic search, restricted to a box around z^* of dimension sufficiently large to ensure that the gradient of any line segment that contains z^* and lies within the box can be calculated accurately.

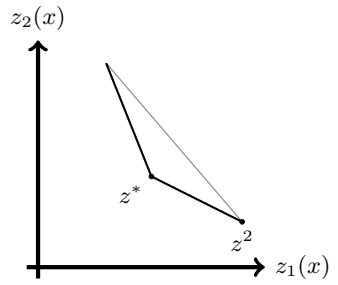
The Boxed Line Method requires the subroutine to produce the three interrelated forms of output. First, it seeks the two endpoints of the line segment, z^1 and z^2. Second, as long as $z^1 \neq z^2$, it seeks the gradient vector, $\vec{w}$, of the line segment $L(z^1, z^2)$. (When $z^1 = z^2$, we say that $\vec{w}$ does not exist). Obviously, if we have z^1 and $z^2 \neq z^1$, then it is trivial to compute $\vec{w}$. However, if $\vec{w}$ is discovered, it is



(a) The NDP z^* dominates all other solutions with the same integer solution: it is an isolated NDP and $z^1 = z^2 = z^*$.



(b) The NDP belongs to exactly one NLS of its slice problem (darkened): the NLS end points are z^1, z^2.



(c) The NDP belongs to exactly two line segments of the slice problem (darkened). In this case, the Line Segment Generation subroutine determines that to the lower right: it takes $z^1 = z^*$, and finds z^2.

Figure 8: Three possible cases for finding the line segment of the slice problem containing z^*.

also possible to find each endpoint by solving the following LPs:

$$x^1 = \arg\min\{z_1(x) : \vec{w}^T z(x) \leq \vec{w}^T z^*, z_2(x) \leq z_2^L, x_I = x_I^*, x \in \mathbb{X}\}, \text{ and} \tag{12}$$

$$x^2 = \arg\min\{z_2(x) : \vec{w}^T z(x) \leq \vec{w}^T z^*, z_1(x) \leq z_1^R, x_I = x_I^*, x \in \mathbb{X}\}. \tag{13}$$

Then $z^1 = z(x^1)$ and $z^2 = z(x^2)$. (By construction, z^1 or z^2 should not be outside the box $B(z^L, z^R)$, since the box's corner points are NDPs.) Therefore, the subroutine either finds *both* endpoints *or* the gradient of the line segment, and then the remaining output can be computed accordingly. Because of this, Algorithm 5, (in the Appendix), giving details of the subroutine, is flexible in the order in which these outputs are discovered.

We note that after the subroutine has found z^1, z^2, and $\vec{w}$ (when it exists), the gradient vector is normalized before it is returned. The reason this is discussed in Section 6.1.

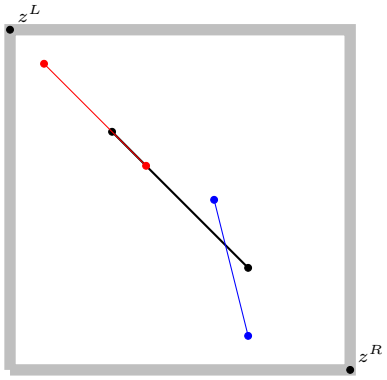
3.5 Complexity of Basic Method

In this section, we prove a worst-case upper bound on the number of IPs solved in order to generate the entire nondominated frontier.We do so by describing the number of IPs solved as a function of the number of nondominated line segments in the frontier.

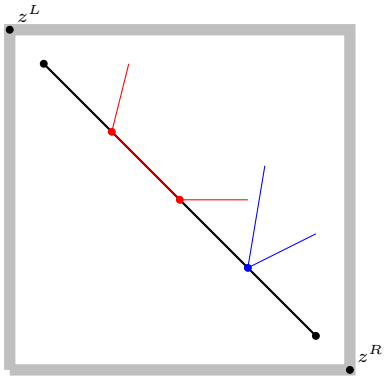
Consider the BOMIP (1) where $\mathbb{X}$ is nonempty and bounded. Recall that $\mathbb{X}_I$ is the projection of $\mathbb{X}$ onto the set of integers, and S is the index set of $\mathbb{X}_I$. Let $S_{\mathcal{N}}$ be the index set of feasible integer solutions whose integer frontiers have nonempty intersection with the nondominated frontier, $\mathcal{N}$, i.e. $S_{\mathcal{N}} = \{s \in S : N^s \cap \mathcal{N} \neq \emptyset\}$. Recall that for all $s \in S_{\mathcal{N}}$, we write $N^s \cap \mathcal{N} = \{L_1^s, L_2^s, ..., L_{n(s)}^s\}$, where, for each $m = 1, \ldots, n(s)$, L_m^s is a (distinct, maximal) line segment in the intersection of the BOMIP NDF and the NDF of the slice problem. We make one technical assumption for ease of exposition: for all distinct pairs of feasible integer solutions, $s, t \in S_{\mathcal{N}}$, we assume that $L_p^s \not\subseteq L_q^t$ and $L_q^t \not\subseteq L_p^s$ for all $p = 1, \ldots, n(s)$ and $q = 1, \ldots, n(t)$. We note that in benchmark problems, (point) intersections between distinct integer frontiers in the nondominated frontier are relatively common, but *containment* of an NLS from one slice problem inside that from another is relatively rare. Such containments do not prevent the algorithm from functioning correctly and returning the entire nondominated frontier, but the counting of NLSs becomes much more complicated. This assumption does not rule out intersections or even *overlap* between integer frontiers; compare Figures 9a and 9b.

In (4), we defined the total number of line segments by $n = \sum_{s \in S_{\mathcal{N}}} n(s)$. For this proof, we use n in the context of a given (nontrivial) box, $Y := B(z^L, z^R)$, where z^L and z^R are NDPs. We take n to be the number of line segments (including isolated NDPs) in the strict interior of Y, which excludes z^L, z^R:

$$n := \sum_{s \in S} |N^s \cap \mathcal{N} \cap Y \setminus \{z^L, z^R\}|. \tag{14}$$



(a) The cases above are "allowed": overlapping and intersecting NLSs from distinct integer frontiers, resulting in $n = 3$.



(b) The cases above are assumed to not occur: a NLS from one slice contained within the NLS from another. Without the red and blue frontiers, $n = 1$ and $g = 2$.

Figure 9: Examples of nondominated frontiers where we assume there is no containment between NLS of distinct integer frontiers, and how to count the number of NLS, n, using (14).

See Figure 9 as an example of counting NLS with this new definition. In addition, let $g \geq 0$ be the number of vertical gaps in the nondominated frontier within Y. Note that since a vertical gap is defined by its two incident NDPs, it must be that $g \leq n + 1$ for all n.

We will compute the specific bounds on the number of IPs solved for different cases of Y with given n and g. Let $\ell(n, g)$ be the *worst case* number of lexicographic optimization IPs solved by the basic method in *completely* processing an arbitrary box $Y = B(z^L, z^R)$ with n nondominated line segments in the strict interior of Y and g vertical gaps in the NDF. By "completely" processing, we mean processing any resulting boxes added to the queue after processing Y, processing any resulting boxes added to the queue after that, and so on until no more (nontrivial) boxes remain in the queue. Similarly, let $s(n, g)$ to be the *worst case* number of single-objective optimization IPs (e.g., scalarized IPs) solved to completely process such a box Y. In general, we assume that no trivial regions Y would appear from the queue because trivial boxes are not added to the queue. Therefore, we define $\ell(0, 0) = s(0, 0) = 0$.

Example 1. *For $n = 0$ and $g = 1$ (see Figure 10), $\ell(0, 1) = 2$ and $s(0, 1) = 0$.*

Proof. Proof of Example 1. For any split $z_2^R < \mu < z_2^L$, we have the first lexicographic optimization IP (5) returns z^R. Since we have $z_2^R < \mu$, the algorithm continues by solving a second lexicographic optimization IP (6), which yields z^L. Note that no new regions are added to the queue (because $B(z^L, z^L)$ and $B(z^R, z^R)$ are trivially small), so we have the iteration of the outer loop terminates having solved $\ell(0, 1) = 2$ lexicographic IPs and $s(0, 1) = 0$ scalarized IPs. ■

Example 2. *For $n = 1$ and $g = 0$ (see Figure 11), $\ell(1, 0) = 1$ and $s(1, 0) = 1$.*

Proof. Proof of Example 2. Since a single nondominated line segment traverses Y, solving the first lexicographic minimization IP (5) with any split line $\mu \in (z_2^R, z_2^L)$ will result in z^* on the split line, i.e., $z_2^* = \mu$. The line generation routine will generate the entire line segment within Y, and because there are no vertical gaps, we must have that $z_2^1 = z_2^L$ and $z_2^2 = z_2^R$. Since z^R is nondominated, the latter implies $z^2 = z^R$, and so z^2 is *closed*. However, z^1 may be open *or* closed (see Figure 11); namely, if $z_1^L < z_1^1$, z^1 will be flagged as open. Regardless of whether it is open or closed, the inner loop will solve the single-objective IP (7) once to confirm that the line segment is indeed nondominated (recall that the IP formulation includes a conditionally strict inequality for the open endpoint which makes z^L infeasible). By our assumption that $n = 1$, the optimal solution must exist on the line segment. At the end of the inner loop, since $z_2^1 = z_2^L$, no other single-objective IPs will be solved by the inner loop since it is known that z^L is the NDP that dominates z^1. After the inner loop concludes, the outer loop would not add any box to the queue, since they would both be trivially small. Thus, $\ell(1, 0) = 1$ and $s(1, 0) = 1$. ■

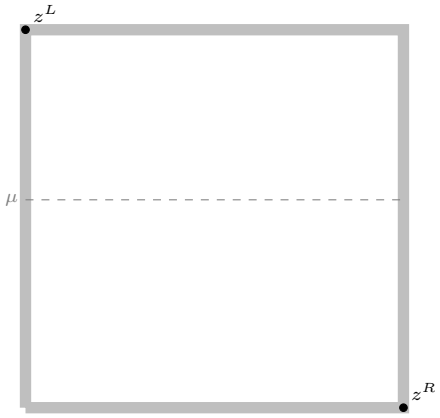


Figure 10: A region Y with $n = 0$ nondominated line segments in its interior and $g = 1$ vertical gaps. An arbitrary horizontal split line is shown as a dashed line. $\ell(0, 1) = 2$ and $s(0, 1) = 0$.

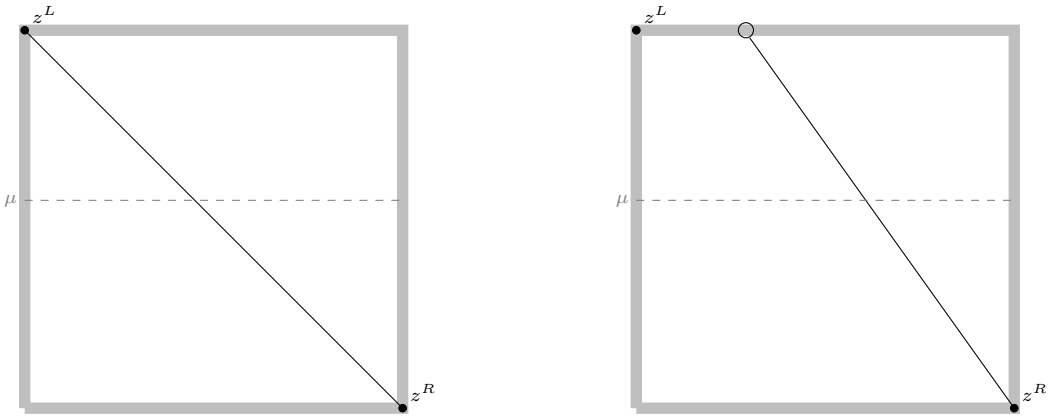


Figure 11: The only two cases for $n = 1$, $g = 0$: The nondominated line segment must be incident to z^R to ensure no vertical gap, but the endpoint z^1 may be either open or closed. Both cases have the same worst-case: $\ell(1, 0) = 1$ and $s(1, 0) = 1$.

As we described while illustrating the basic method, every iteration of the outer loop identifies either a nondominated line segment or a vertical gap. The following theorem proves that the algorithm eliminates that line segment or gap *completely* before adding two new boxes to the queue. This is an important characteristic which, for example, does not apply to the Triangle Splitting Algorithm since it may divide a single nondominated line segment into several smaller segments over more than one iteration. We next formalize and prove this unique characteristic, which will allow us to provide an upper bound for the number of iterations of the outer loop as well as an appropriate equation for $\ell(n, g)$.

Theorem 1. *Suppose an iteration of the outer loop begins with a region Y having $n \geq 0$ nondominated line segments in its strict interior and $g \geq 0$ vertical gaps where $n + g \geq 1$.*

1. *At the end of the iteration, (up to) two nontrivial regions Y', Y'' are added to the queue, say with n', n'' numbers of nondominated line segments in their interior and g', g'' numbers of vertical gaps, respectively. Then we have $n' + n'' \leq n$ and $g' + g'' \leq g$ where **at least one** inequality is strict. That is, after each iteration of the outer loop, one NLS or one vertical gap is eliminated completely from the total unexplored region(s).*
2. *The number of iterations of the outer loop required to finish completely processing Y (including all subsequent boxes) is bounded above by $n + g$.*
3. *$\ell(n, g) = n + 2g$.*

Proof. Proof of Theorem 1 (1). In one iteration of the outer loop, there are two cases for the split line μ: either it intersects a vertical gap or it intersects a nondominated line segment of the nondominated frontier.

In the first case, when the split line μ intersects a vertical gap of the nondominated frontier, we have by construction that the two lexicographic minimizations (5) and (6) identify the two NDPs incident to the vertical gap, $\hat{z}$ (above) and z^* (below). Let the two resulting regions be $Y' = B(z^L, \hat{z})$ and $Y'' = B(z^*, z^R)$. We have of course decreased the total number of vertical gaps in the frontier(s), so $g' + g'' < g$. However, the total number of nondominated line segments in the interiors of Y' and Y'' is no greater than n, i.e. $n' + n'' \leq n$. We have satisfied the claim in first case.

In the second case, z^* belongs to some NLS $L(z^1, z^2)$. Then we initialize regions Y' and Y'' based on NDPs – either the closed endpoints (i.e., z^1, z^2) or the NDPs that dominate the open endpoints (i.e., $\hat{z}^1, \hat{z}^2$) – discovered by the inner loop in such a way that guarantees the single nondominated line segment $L(z^1, z^2)$ is not split. Hence, we must have one fewer nondominated line segment in the remaining regions, i.e. $n' + n'' < n$. We also have $g' + g'' \leq g$, which satisfies the claim in the second case. ■

Proof. Proof of Theorem 1 (2). This result follows trivially from Theorem 1 (1). Since every iteration of the outer loop eliminates an NLS or a vertical gap from the unexplored region(s), then after $n + g$ iterations, there will only be trivial unexplored regions remaining. ■

Proof. Proof of Theorem 1 (3). This result follows from the proof of Theorem 1 (1). Only the outer loop solves lexicographic IPs. In every iteration of the outer loop, the horizontal split line μ either intersects a vertical gap in the ND frontier or it intersects a nondominated line segment. If the split line intersects a nondominated line segment, one lexicographic IP is solved for z^*, and if the split line intersects a vertical gap, two lexicographic IPs are solved for z^* and $\hat{z}$. Then it takes at most $\ell(n, g) = n + 2g$ lexicographic IP solves to process Y (at most one for each NLS and at most two for each vertical gaps). Also note that Examples **??** and 2 satisfy this upper bound. ■

Lemma 1. *For all $g \in \{0, 1, 2\}$, $s(1, g) = 1$.*

Proof. Proof of Lemma 1. Note that we have already shown $s(1, 0) = 1$ by Example 2. Therefore, suppose $g = 1$ or $g = 2$.

First, consider when the split line μ intersects the nondominated line segment of the frontier. Then one lexicographic IP (5) finds z^* such that $z_2^* = \mu$ (we do not count lexicographic IP solves). Once the line segment containing z^* is generated, the single-objective IP (7) is solved to determine whether the line segment is dominated by another NDP. As mentioned previously, the algorithm checks if $z_2^1 = z_2^L$ and $z_1^L < z_1^1$, in which case endpoint z^1 is flagged as open, and it is known that z^L dominates z^1 (and similarly for z^2 and z^R). Note that by assumption, $n = 1$ implies there are no other NDPs in the interior that dominate the line segment, and by construction we have the conditionally strict inequality that prevents from finding z^L (or z^R) in the case that z^L dominates z^1 (or z^R dominates z^2). Hence, the solution to the single-objective IP (7) will certainly map to a point on the line segment, which will terminate the while loop within the inner loop. Since the NDPs that dominate the open endpoints are known, no additional single-objective IPs will be solved to discover what points dominate z^1 or z^2, and so the inner loop terminates.

Once the inner loop has generated the nondominated line segment within the given region, the outer loop will add at most two nontrivial boxes, whose interiors contain $n' = n'' = 0$ nondominated line segments, to the queue. So we have for some nonnegative integers $(g', g'') \in \mathbb{Z}_+^2$ where $g' + g'' = g$ but $g', g'' \leq 1$ (this follows from $g' \leq n' + 1$ and $g'' \leq n'' + 1$ where $n' = n'' = 0$). Therefore, $s(1, g) = 1 + s(0, g') + s(0, g'') = 1$ because $s(0, 0) = 0$ by definition, and $s(0, 1) = 0$ by Example **??**. Thus, $s(1, g) = 1$ for $g \in \{0, 1, 2\}$.

Second, consider when the split line μ crosses at a vertical gap in the frontier. Note that a single-objective IP is not solved until the inner loop is initiated, which respectively does not occur until a solution is found on the split line. Therefore there can be at most g iterations of the outer loop in which $z_2^* < \mu$, wherein each iteration is only performing lexicographic IP solves as the split lines identify vertical gaps in the frontier which are then removed by initiating new boxes to the queue (note we do not count any of these lexicographic IP solves). Therefore, in at most g iterations the lexicographic optimization IP will return z^* such that $z_2^* = \mu$. At such point, the process would follow the process argued above, so again we have $s(1, g) = 1$ for $g \in \{0, 1, 2\}$. ■

Since $g \leq n+1$, we can therefore define the worst case number of IP solves as a function of only n, the number of NLS in the strict interior of Y:

$$\hat{\ell}(n) = \max_{g=1..n+1} \ell(n,g), \tag{15}$$

$$\hat{s}(n) = \max_{g=1..n+1} s(n,g). \tag{16}$$

We so far have $\hat{s}(0) = 0$, $\hat{s}(1) = 1$, and $\hat{\ell}(n) = 3n+2$ for all $n \geq 0$ (since $g \leq n+1$ gives $\ell(n,g) = n+2g \leq n+2(n+1) = 3n+2$).

Lemma 2. *For all $n \geq 2$, $\hat{s}(n) = n+2+\hat{s}(n-1)$.*

Proof. Proof of Lemma 2. Assume there are $n \geq 2$ nondominated line segments. Without loss of generality, we suppose the number of vertical gaps in the frontier is arbitrary and only consider the case that the split line μ intersects a nondominated line segment; otherwise, when the split line intersects a vertical gap, the box is processed in the outer loop without the solution of any scalarized or single objective IPs. Once the lexicographic IP (5) (which is not counted for $\hat{s}$) finds solution z^* on the split line, the inner loop is initiated. Since there are $n-1$ other nondominated line segments in the box, there are at worst $n-1$ scalarized IPs (7) solved in updating the endpoints z^1 and z^2. The nth solve of scalarized IP (7) confirms that the final endpoints on the line segment indeed define a nondominated line segment. The worst case is when both z^1 and z^2 are open, $z_2^1 < z_2^L$, and $z_1^2 < z_1^R$ since two additional single-objective IP solves are required to find the NDPs that dominate z^1 and z^2 (see Figure 12). Therefore, we have for some $n' + n'' = n-1$, $\hat{s}(n) = n+2+\hat{s}(n')+\hat{s}(n'')$. So it follows that the worst case is $\hat{s}(n) = n+2+\max_{i=0,..,n-1}\{\hat{s}(i)+\hat{s}(n-1-i)\}$.

We now use induction to complete the proof. First observe that for $n=2$, as required,

$$\hat{s}(n) = n+2+\max_{i=0,1}\{\hat{s}(i)+\hat{s}(2-1-i)\} = n+2+\hat{s}(1)+\hat{s}(0) = n+2+\hat{s}(1),$$

since $\hat{s}(0) = 0$. Now make the inductive assumption that for some $m \geq 2$, $\hat{s}(n) = n+2+\hat{s}(n-1)$ for all $n = 2,\ldots,m$, and consider $\hat{s}(m+1) = m+3+\max_{i=0,..,m}\{\hat{s}(i)+\hat{s}(m-i)\}$. The case that $i=0$ (and $i=m$) in the max term gives $\hat{s}(0)+\hat{s}(m) = \hat{s}(m)$ since $\hat{s}(0)=0$. The case that $i=1$ (and $i=m-1$) in the max term gives $\hat{s}(1)+\hat{s}(m-1) = 1+\hat{s}(m-1)$ since $\hat{s}(1)=1$. But, by the inductive assumption $\hat{s}(m) = m+2+\hat{s}(m-1) > 1+\hat{s}(m-1)$, since $m \geq 2$. So the case $i=1$ (and $i=m-1$) cannot achieve the maximum. Finally, again by the inductive assumption,

$$\begin{aligned}
\max_{i=2,\ldots,m-2}\{\hat{s}(i)+\hat{s}(m-i)\} &= \max_{i=2,\ldots,m-2}\{i+2+\hat{s}(i-1)+m-i+2+\hat{s}(m-i-1)\} \\
&= m+4+\max_{i=1,\ldots,m-3}\{\hat{s}(i)+\hat{s}(m-2-i)\} \\
&\leq m+4+\max_{i=0,1,\ldots,m-3,m-2}\{\hat{s}(i)+\hat{s}(m-2-i)\} \\
&= m+4+\hat{s}(m-1)-(m-1+2) \\
&= 3+\hat{s}(m-1).
\end{aligned}$$

But $\hat{s}(m) = m+2+\hat{s}(m-1) > 3+\hat{s}(m-1)$ since $m \geq 2$, so none of the cases $i = 2,\ldots,m-2$ can achieve the maximum. We conclude that $\max_{i=0,..,m}\{\hat{s}(i)+\hat{s}(m-i)\} = \hat{s}(m)$ and so $\hat{s}(m+1) = m+3+\hat{s}(m)$, as required. ■

Proposition 1. *For all $n \geq 1$, $\hat{s}(n) = \frac{n(n+1)}{2} + 2(n-1)$.*

Proof. Proof of Proposition 1. We again use induction. The case $n=1$ follows since $\hat{s}(1) = 1 = \frac{1(1+1)}{2} + 2(1-1)$. Now assume that, for some $n \geq 1$, $\hat{s}(n) = \frac{n(n+1)}{2} + 2(n-1)$, and consider $\hat{s}(n+1)$. By Lemma 2 and the inductive assumption,

$$\begin{aligned}
\hat{s}(n+1) = n+3+\hat{s}(n) = n+3+\frac{n(n+1)}{2}+2(n-1) &= \frac{1}{2}n^2+\frac{7}{2}n+1 \\
= \frac{1}{2}n^2+\frac{3}{2}n+1+2n &= \frac{(n+1)(n+2)}{2}+2n,
\end{aligned}$$

as required. ■

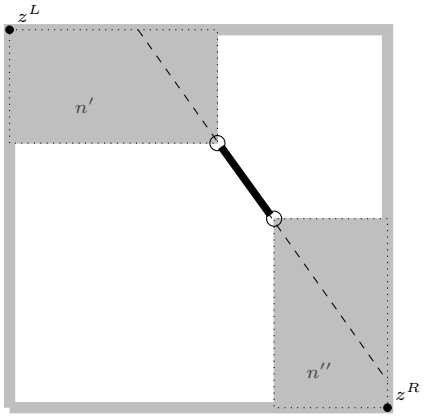


Figure 12: General frontier generated where there are $n \geq 1$ nondominated line segments. In the worst case for counting single-objective IP solves, both endpoints are open.

4 Recursive Method

There is a natural way to make the Boxed Line Method a recursive method, which has substantially better worst-case complexity. The fundamental concept is to transform the inner loop into a recursive procedure that either confirms that a line segment is nondominated or identifies an NDP that dominates it. Whenever such an NDP is found, the inner loop is called recursively to find its nondominated line segment. In addition to modifying the inner loop, we must introduce a line segment *trimming* subroutine and modify the outer loop to fit the recursive paradigm. The recursive method is summarized in Figure 15.

The initialization stage remains the same, and the outer loop follows the same procedure as the basic method before calling the inner loop, i.e. it selects boxes from the queue, chooses a split line μ, solves lexicographic minimization (5), and follows the BBM procedure when the NDP found is off the split line. When the outer loop has chosen box $B(z^L, z^R)$ from the queue and finds NDP z^* on the split line, i.e. $z_2^* = \mu$, the outer loop then calls the recursive inner loop for the first time, which we call *depth level 0*.

4.1 Recursive Inner Loop

The recursive inner loop takes as input: a box bounded by two NDPs, $B(z^L, z^R)$; the NDP z^* whose NLS it seeks to generate; and the *set* of all line segments "inherited" from its parent calls, which we call $\mathcal{L}$. On the first call of the inner loop (depth level 0), the set $\mathcal{L}$ is empty; for greater depth levels, this set will be nonempty and will be used for the line segment trimming subroutine. The output from the recursive function is a set of nondominated line segments, M, including the one that contains z^*.

We now describe the steps of the recursive function.

It first calls the line segment generation subroutine, as in the basic method. If z^* is an isolated NDP, then the recursive function terminates and returns the isolated NDP. Otherwise, the subroutine returns line segment $L(z^1, z^2)$ and its gradient vector $\vec{w}$.

Its second step is to call the line segment trimming subroutine, which "trims" the current line segment so that its endpoints do not exceed any of the previously-found line segments. This step prevents cycling. We postpone further discussion of this feature to Section 4.2. For now, we note that there is no trimming to be done at depth level 0 since $L(z^1, z^2)$ is the first line segment that has been generated in the box $B(z^L, z^R)$. Once trimming is complete, the same checks as in the basic method's inner loop are made, to see if either endpoint is dominated by z^L or z^R (and, if so, to update the endpoint as open, accordingly). The resulting line segment is then added to the set of line segments, $\mathcal{L}$, for future line trimming.

The next step in the recursive inner loop is to solve the scalarized IP

$$z(y^*) = \min\{\vec{w}^T z(x) \ : \ z_1(x) \leq z_1^2, \ z_2(x) \leq z_2^1, \ x \in \mathbb{X}\}, \tag{17}$$

where each inequality is strict if the corresponding endpoint is open, to obtain solution y^*. There are two cases. (1) If $L(z^1, z^2)$ is nondominated, then $z(y^*)$ lies on the line segment, i.e. $\vec{w}^T z(y^*) = \vec{w}^T z^*$. This is the stopping criterion for the while loop. When it is satisfied, the recursive function terminates and returns the nondominated line segment, together with all others it found, stored in M. (2) The more

interesting case is when $\vec{w}^T z(y^*) < \vec{w}^T z^*$, i.e., when $L(z^1, z^2)$ is *dominated* by $z(y^*)$. In this case, the function recurses and calls itself, at the next depth level. Assume $z(y^*)$ is to the left of z^*; the case of $z(y^*)$ to the right can be handled similarly. Then the following is the input to the recursive call:

- $B(\alpha, z^*)$, the new box, where α is the NDP from $M \cup \{z^L\}$ closest to $z(y^*)$ on the left (note that z^* is the closest known NDP to the right of $z(y^*)$);
- $z(y^*)$, the new NDP (it is the NLS containing it that the function seeks); and
- $\mathcal{L}$, the set of line segments, used for line segment trimming.

The choice of $B(\alpha, z^*)$ as the new box prevents re-discovery of line segments already found, and ensures that only the portion of the line segment containing $z(y^*)$ currently not known to be dominated is explored in the recursive call. Note that when M is empty, $\alpha = z^L$.

After all deeper levels of recursive processing are complete, a set of nondominated line segments, M, is returned to the current depth level call of the recursive inner loop. Then the recursive function updates the line segment $L(z^1, z^2)$. In doing so, it follows the same rules as in the basic method's inner loop. However, instead of v found from the LP (8), it chooses the NDP from M that is the nearest to and strictly to the left of z^*. That is, it chooses

$$p = \text{argmin}\{z_2 : z \in M, z_1 < z_1^*\}. \tag{18}$$

The coordinate p_2 gives the updated z_2^1; the updated z_1^1 value is calculated to ensure z^1 remains on its original line segment. Whether it is closed or open is determined by whether or not p lies on the line segment. Since p is guaranteed to be an NDP, there is no need to solve a separate LP to discover the NDP that dominates z^1 when it is open.

Once the endpoint z^1 (or z^2, if exploring to the right of z^*) is updated, the while loop again solves the scalarized IP (17). If the updated line segment is nondominated, then the stopping criteria is met and the line segment $L(z^1, z^2)$ is returned, along with any others collected in M. Otherwise, the while loop continues.

The recursive inner loop accumulates all nondominated line segments identified throughout all depths of recursion. This accumulated set, M, is returned to the outer loop, which updates the nondominated frontier and queue accordingly (details are in Section 4.3).

4.2 Line Segment Trimming Subroutine

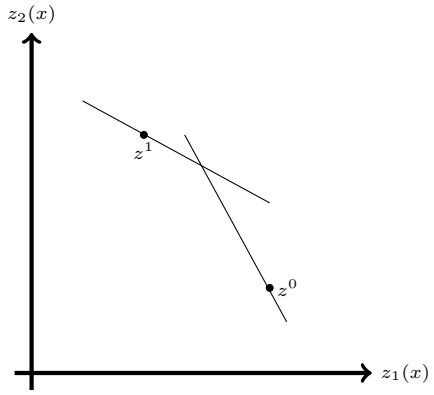
To motivate this subroutine, we give an example of how – without it – cycling can occur. Consider two intersecting slices from distinct integer solutions, as in Figure 13a. Suppose z^0 were first found by the split line and lexicographic minimization. Then line generation would generate its full line segment, L_0, and scalarization by its gradient $\vec{w}^0$ would result in finding z^1 (recall that the scalarized IP (7) is constrained by its endpoints). For the next level of recursion, line generation would generate the full line segment containing z^1, L_1, and its gradient $\vec{w}^1$. Note, however, that L_1 intersects L_0, and in fact the lower right endpoint of L_1 is dominated by z^0. Thus scalarization by $\vec{w}^1$ would yield z^0 again, and the algorithm would cycle.

In general, there is a risk of cycling whenever a call to the recursive inner loop generates a line segment that intersects the line segment of a parent call. We introduce the following simple routine to detect such intersections and update the child's line segment so as to prevent cycling.

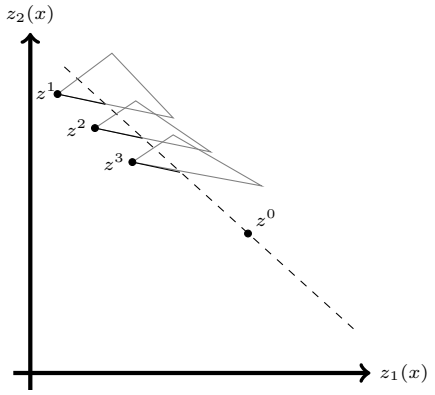
Let $L_\alpha := L(\alpha^1, \alpha^2)$ denote the current line segment, which is generated from NDP z^α, and has gradient vector $\vec{w}^\alpha$. Suppose that a line segment from a parent call, denoted by a $L_\beta := L(\beta^1, \beta^2)$, generated from the NDP z^β and has gradient $\vec{w}^\beta$. The two line segments L_α and L_β can be extended to lines that intersect provided $\vec{w}^\alpha \neq \vec{w}^\beta$ (recall the gradient vectors are normalized). Their point of intersection, $\gamma \in \mathbb{R}^2$, is easily computed: it is the solution of the system of two linear equations given by $\vec{w}^\alpha \gamma = \vec{w}^\alpha z^\alpha$ and $\vec{w}^\beta \gamma = \vec{w}^\beta z^\beta$. The resulting intersection point (γ) is contained in both line segments if and only if

$$\gamma_1 \in [\alpha_1^1, \alpha_1^2] \cap [\beta_1^1, \beta_1^2] \quad \text{and} \quad \gamma_2 \in [\alpha_2^2, \alpha_2^1] \cap [\beta_2^2, \beta_2^1]. \tag{19}$$

If γ satisfies (19), then the current line segment, L_α, is trimmed. Exactly one of the two subsets of L_α, $L(\alpha^1, \gamma)$ or $L(\gamma, \alpha^2)$, must be dominated by L_β. Also $z^\alpha \in L_\alpha$ is an NDP. Thus L_α is trimmed so as to retain the portion of it that contains z^α: if $\gamma_1 < z_1^\alpha$, then update $\alpha^1 = \gamma$ and flag α^1 to be closed; otherwise, if $\gamma_1 > z_1^\alpha$, then update $\alpha^2 = \gamma$ and flag α^2 to be closed. In the unlikely case that $\gamma_1 = z_1^\alpha$, L_α is trimmed as follows: if $\gamma_1 < z_1^\beta$, then update $\alpha^2 = \gamma$ and flag α^2 to be closed; otherwise, if $\gamma_1 > z_1^\beta$, then update $\alpha^1 = \gamma$ and flag α^1 to be closed. Note that updating the appropriate endpoint of L_α to be



(a) Without the line segment trimming, the recursive method would cycle between finding z^0 and z^1.



(b) The recursive method first finds NDP z^0 and its line segment (dashed). By recursing, it finds z^1, z^2, and z^3 in that order, where the line segments after trimming are darkened.

Figure 13: The line segment trimming subroutine prevents the recursive method from cycling.

the intersection point γ and flagging it as closed prevents rediscovery of any point in L_β that dominates any part of L_α.

The line trimming process must be repeated for all line segments that have been inherited by the current call. For instance, observe that in Figure 13b, if all subsequent line segments were not trimmed by the line segment containing z^0 then a cycle would occur.

4.3 Outer Loop Modification

The only major modification in the outer loop is the handling of the output from the recursive inner loop, which may return discontinuous portions of the nondominated frontier (see Figure 14). Therefore, we must allow the outer loop to add more than one NLS to the nondominated frontier and more than two boxes to the queue, as necessary.

The latter requires a simple check of neighboring nondominated line segments. Consider checking $L_1 := (z^1, z^2)$ which is immediately to the left of $L_2 := (z^3, z^4)$. When $z_1^2 < z_1^3$ and $z_2^2 > z_2^3$, the outer loop should add the box $B(z^2, z^3)$ to the queue. Note that we must also consider the boundary NDPs, z^L and z^R, while doing this check. In Figure 14, four boxes are added to the queue because four pairs of adjacent NLSs satisfy the criteria. One pair of NLSs does not satisfy the criteria: the isolated NDP z^3 and the NLS containing z^*.

4.4 Complexity of Recursive Method

Let $\hat{\ell}_R(n)$ be the worst case number of lexicographic IPs solved by the *recursive* method in completely processing an arbitrary box $B(z^L, z^R)$ with n NLSs (and an arbitrary number of vertical gaps), and let $\hat{s}_R(n)$ be the same but for scalarized IPs. Recall that the outer loop only solves lexicographic IPs, and the recursive inner loop only solves scalarized IPs.

The initialization stage and most of the functionality of the outer loop is unchanged, i.e., in every iteration the outer loop solves one lexicographic IP to identify a NLS, or it solves two lexicographic IPs to identify a vertical gap. What has changed is the updating procedure for the nondominated frontier and queue after the recursive inner loop has terminated. Regardless, the upper bound for the basic method's number of lexicographic IP solves is valid for the recursive method, i.e. $\hat{\ell}_R(n) = \hat{\ell}(n) = 3n + 2$. However, we must reconsider an upper bound for the number of scalarized IP solves.

Proposition 2. *For all $n \geq 1$, $\hat{s}_R(n) = 2n - 1$.*

Proof. Proof of Proposition 2. Let z^L, z^R be the corner points for a box with $n \geq 1$ NLSs and suppose that frontier in the box contains no vertical gaps. This assumption can be made without loss of generality, since vertical gaps in the frontier are only discovered in the outer loop, by a lexicographic IP solve that

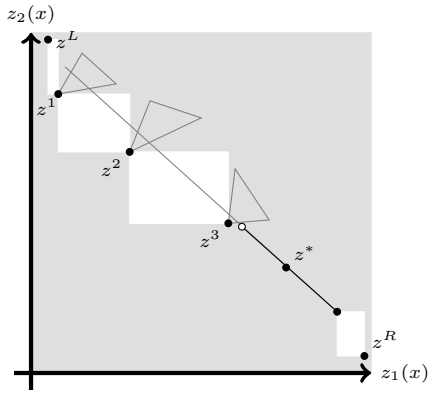


Figure 14: Both the recursive method and the multibox method (Section 5.1) are likely to return discontinuous portions of the nondominated frontier (e.g., the NLS containing z^* and 3 isolated NDPs in the example above), in which case the algorithm adds more than two boxes to the queue (in white).

yields an NDP not on the split line; they do not change the number of scalarized IPs that need to be solved. Thus we may safely assume that the NDF in the box consists of a vertically continuous part of the frontier.

Since there are no vertical gaps, any split line intersects an NLS, containing the NDP z^* found by the lexicographic IP. Then the line segment in a slice containing z^* is determined (using LPs) and the recursive inner loop is initiated.

In the recursive inner loop, there are two roles for scalarized IPs. Type I confirms that the current line segment is nondominated, and every nontrivial line segment requires exactly one of these IPs to be solved. Isolated NDPs require none, so in the worst case, all NLSs have dimension 1, and one Type I scarized IP is solved per NLS in the frontier. Type II identifies an NLS for the first time: it shows that a portion of the current line segment is dominated, by returning a new NDP, and the recursive call returns the NLS containing the new NDP. Within the recursive call, it is impossible to rediscover the same NLS. Therefore, it will take solution of no more than $n-1$ Type II scalarized IPs to discover all of the NLSs (the first NLS must be found by a lexicographic IP). Since all n NLSs require solution of a Type I scalarized IP to confirm nondominance, a total of $\hat{s}_R(n) = n + (n-1) = 2n-1$ scalarized IPs will be solved, in the worst case. ■

5 Enhancements to the Basic Method

We propose two enhancements to the basic method (given in Section 3) that have the potential to improve its computational performance.

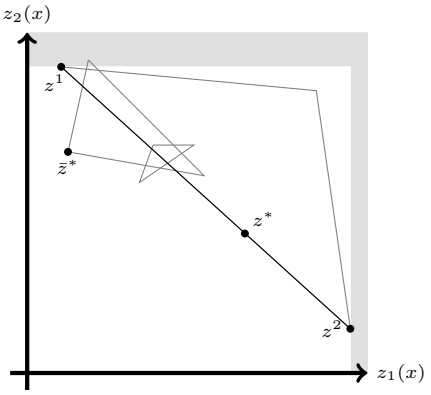
The first seeks to rectify a shortcoming of the basic method: many NDPs found by solution of a scalarized IP are "forgotten"; only those that induce an endpoint of the current NLS are "remembered", as they form a corner point of a new box. Thus we propose a modification that makes use of all NDPs found in the course of determining the endpoints of the current NLS. This results in the addition of multiple boxes (not just two) to the queue at the completion of the inner loop, hence we call this the *multi-box* variant.

The second exploits the observation that when both corner points of a box have the same integer part of their solution, (they belong to the same slice), there is a good chance that the NDF within the box is precisely the part of the slice within the box.

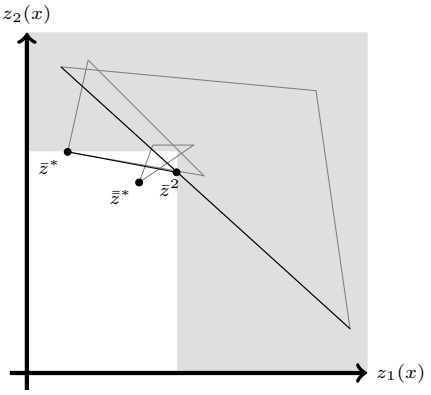
We discuss each of these, in turn, below.

5.1 The Multi-Box Method

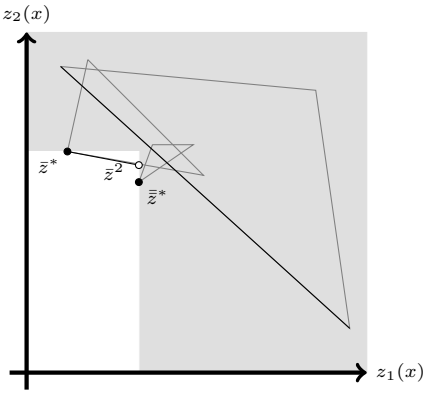
The inner loop proceeds in the same manner as in the basic method, except that every NDP found by a scalarized IP is stored in a set, $\bar{M}$, which is returned along with the final NLS containing z^*. The outer loop then processes the points in $\bar{M}$ together with the corner points of the current box, in order, adding a new box to the queue for each consective pair of points, which form the corner points of the new box.



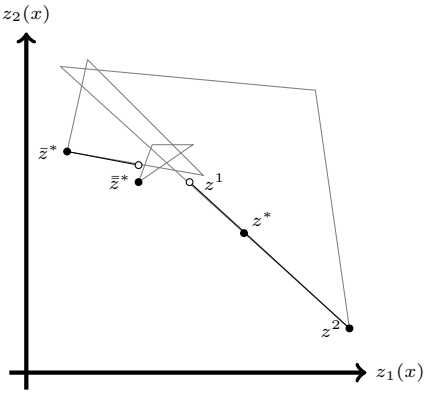
(a) Level 0: The original NDP is z^*, and line generation returns $L_0 = L(z^1, z^2)$, which is not trimmed. Solving a scalarized IP over the white region identifies NDP $\bar{z}^*$.



(b) Level 1: The line generated for $\bar{z}^*$ is trimmed by its intersection with L_0, resulting in $L_1 = L(\bar{z}^*, \bar{z}^2)$ with $\bar{z}^2$ closed. NDP $\bar{\bar{z}}^*$ solves the next scalarized IP. Level 2: Returns isolated NDP $\bar{\bar{z}}^*$.



(c) Level 1: Update endpoint $\bar{z}^2$, which is now open. One more scalarization over the white region identifies that $L(\bar{z}^*, \bar{z}^2)$ is nondominated.



(d) Level 0: Update endpoint z^1 with respect to $p = \bar{\bar{z}}^*$, so z^1 is now open. Solving one final scalarized IP confirms that $L(z^1, z^2)$ is nondominated.

Figure 15: The recursive inner loop applied to NDP $\bar{z}^*$ returns nondominated line segments $L(z^1, z^2)$ and $L(\bar{z}^*, \bar{z}^2)$ and the isolated NDP $\bar{\bar{z}}^*$.

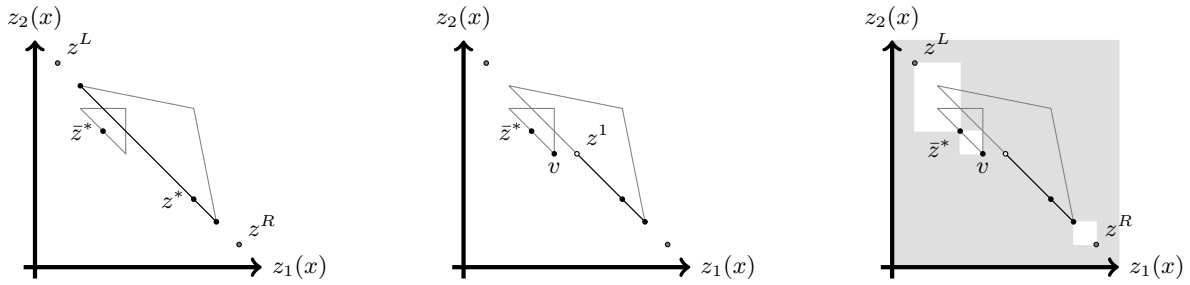
Of course, the consecutive points that are either endpoints or induce endpoints of the NLS containing z^* are skipped, as within the box they form, the NDF is already known to be this NLS.

The method may be illustrated using the example given in Figure 14. Suppose that the inner loop finds z^1, z^2, z^3, in that order. In the multi-box method, these are returned by the inner loop, together with the line segment containing z^*. The outer loop then processes the points z^L, z^1, z^2, z^3, the unmarked endpoint at the right end of the NLS containing z^* and z^R, adding the four white boxes to the queue.

This multi-box enhancement allows the knowledge of the NDPs found by scalarized IPs in the inner loop to be exploited; solution of additional IPs, which the original basic method would need to "re-find" the NLSs containing these NDPs, is avoided. However, this comes at a cost: the multi-box variant forfeits the property that every box added to the queue has corner points that are *endpoints* of NLss. It may split a single NLS into multiple NLSs.

Figure 16 illustrates this issue. Here the NDP found by scalarization in the inner loop is not necessarily an endpoint of an NLS. This situation can arise when slices have parallel line segments. Therefore, once the outer loop adds the two boxes, $B(z^L, \bar{z}^*)$ and $B(\bar{z}^*, v)$, to the queue, the single NLS is split into two, which means that roughly twice as much work must be done to find both smaller line segments. Thus we offer no complexity analysis for the multi-box method.

There is a second issue for the enhanced, multi-box, method that was not a problem for the basic method. Recall that in the basic method, if $L(z^1, z^2)$ is the result of the line generation routine, then



(a) In $B(z^L, z^R)$, the NDP z^* is found on the split line, and its full line segment (darkened) is generated. Scalarization by its gradient has multiple optimal solutions, but only NDP $\bar{z}^*$ is returned.

(b) The minimization to find v is used to update z^1, which is now open. One more scalarization confirms that the darkened line segment is nondominated. Solving LP (10) finds that v is the NDP that dominates z^1.

(c) The enhanced, multi-box, outer loop then adds 3 boxes to the queue (in white). Notice that the single nondominated line segment containing $\bar{z}^*$ is split into two separate boxes.

Figure 16: In some cases, the outer loop for the enhanced method will create boxes whose corner points are not endpoints of a nondominated line segment.

the basic inner loop checks if z^1 is dominated by z^L, and if so, flags z^1 as open. Then the scalarized IP is solved with the conditionally strict inequality, which prevents it from finding z^L. In the case that z^L is *not* contained in an NLS that extends into the current box, flagging z^1 as open saves a scalarized IP solve. However, in the case that z^L *is* contained in an NLS that extends into the current box, flagging z^1 as open when it is dominated by z^L may cause the multi-box method to solve one extra lexicographic IP and one extra scalarized IP. It may also report two consecutive line segments in the NDF where one will suffice.

To see how this can occur, refer to Figure 17. Here, with z^1 flagged as open, the first scalarized IP finds $z(y^*)$ very near[1] Since the multi-box method would add the box $B(z^L, z(y^*))$ to the queue, it will expend one extra lexicographic and one extra scalarized IP solve to process the box, and report one extra (small) NLS instead of one full-length NLS. Flagging z^1 (z^2) as closed from the beginning of inner loop, regardless of whether or not it is dominated by z^L (z^R) solves this problem. With z^1 flagged as closed, we expect the first scalarized IP to return z^L. Then the inner loop identifies the end of its line segment, v, which is proved to be an NPD, and the single box $B(z^L, v)$ is added to the queue. Even if numerical issues in the solution of the first scalarized IP lead it to return $z(y^*) \neq z^L$, with the correct tolerances set in the IP solver, we expect $z_2(y^*)$ to be within ϵ of z_2^L, and so $B(z^L, z(y^*))$ would not be added to the queue, since it is trivially small.

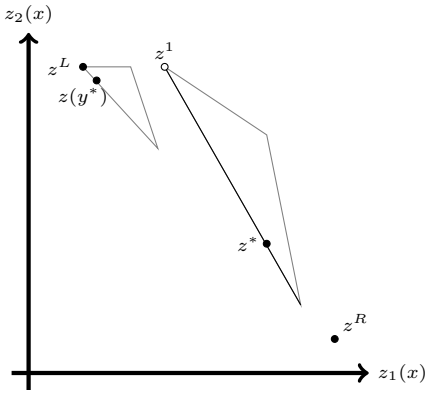
Thus, in the multi-box method, the endpoints of the generated line segment containing z^* are always flagged as closed, even if they are dominated by a current box corner.

5.2 Same Integer Solution (SIS) Enhancement

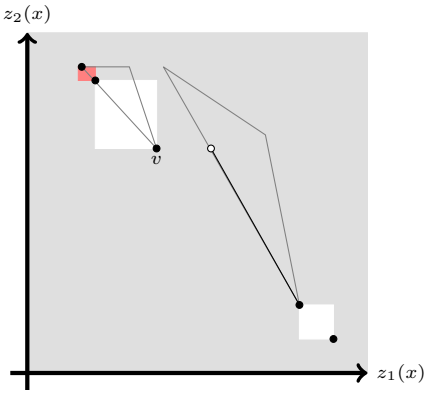
Benchmark instances used in previous work on BOMIP algorithms, (such as Boland et al. (2015b) and Soylu and Yıldız (2016)), have NDFs consisting of a relatively small number of continuous sections, generated by relatively few slices, each having many, small NLSs. Figure 18 gives plots of two such instances. In one, only 5 slices contribute to the NDF, which consists of only 4 continuous sections. In the other, 15 slices yield the NDF consisting of 10 continuous sections. Each continuous section has several NLSs from the same slice. These two instances are typical of the benchmark instances used in prior work.

For these instances, when both corner points of a box have the same integer solution, i.e., when $z^L = z(x^L)$ and $z^R = z(x^R)$ where $x_I^L = x_I^R =: x_I^*$, it is very likely that the frontier between z^L and z^R is precisely that of the slice problem for integer solution, x_I^*. The *Same Integer Solution (SIS)* enhancement exploits this observation. In the case that the slice problem does yield the NDF in the current box, the

[1]Recall that the conditionally strict inequality is shorthand for implementing a shift by $\epsilon > 0$ to z^L. Thus the NDP found will be about ϵ away from z^L.



(a) If z^1 were to be flagged as open when it is dominated by z^L, then the scalarized IP would find NDP $z(y^*)$ very near to z^L.



(b) The enhanced, multi-box, method would then add $B(z^L, z(y^*))$ (in red) to the queue, even though it is very small.

Figure 17: The enhanced, multi-box, method treats both endpoints as closed, even if they are dominated by z^L or z^R, since treating them as open (as in the basic and recursive methods), may lead to the unnecessary creation and processing of an additional box, as shown here in red.

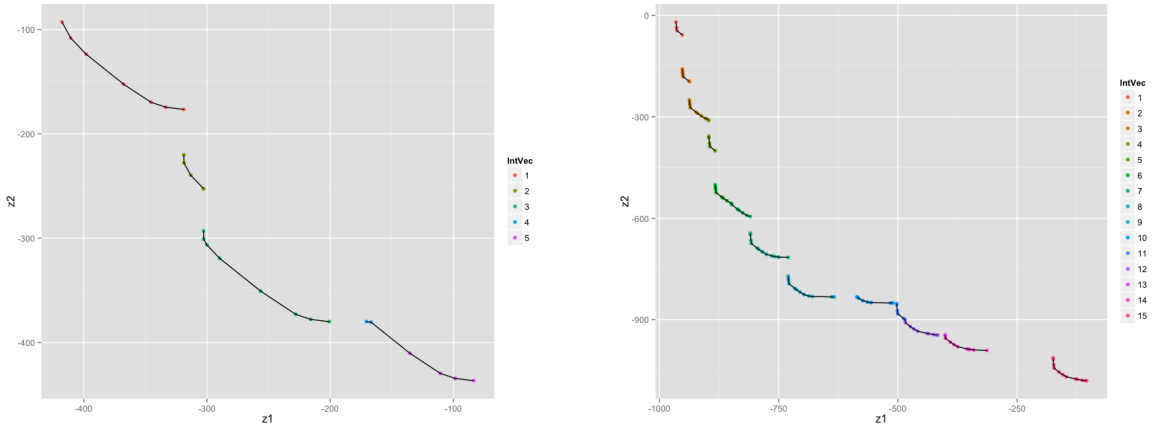


Figure 18: The nondominated frontiers for two benchmark instances. The keys indicate the integer vector associated with each NDP. Notice the similar structure in both: each integer vector contributes several small and continuous nondominated line segments to the frontier.

SIS enhancement may need to solve only one scalarized IP, having one no-good (sometimes called "tabu") constraint, instead of one scalarized IP for each NLS; once it is determined that the slice problem yields the frontier, LPs, rather than IPs, may be used to find all NLSs.

When the corner points of a box are generated by a single integer solution, x_I^*, the frontier of its slice problem must be entirely on or below the line segment $L(z^L, z^R)$. (This follows from convexity of the image of the LP feasible set in objective space.) Figure 19 shows examples of such slice problem frontiers. Let $\vec{w}^t$ denote the gradient vector of $L(z^L, z^R)$.

The following scalarized IP with respect to $\vec{w}$ includes a no-good constraint, $x_I \neq x_I^*$, which makes all points from the slice of x_I^* infeasible:

$$\min\{\vec{w}^T z(x) : z_1(x) \leq z_1^R, \quad z_2(x) \leq z_2^L, \quad x_I \neq x_I^*, \quad x \in \mathbb{X}\}. \tag{20}$$

(We assume that the no-good constraint is implemented linearly in the usual way.) Let y^* be an optimal solution to (20). Because of the no-good constraint, $z(y^*)$ is not necessarily nondominated. However, we make a simple observation: if the point $z(y^*)$ is on or above the line segment $L(z^L, z^R)$, i.e. $\vec{w}^T z^L \leq$

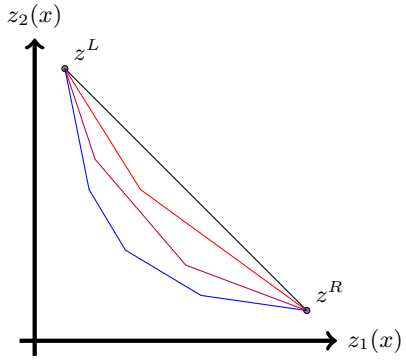


Figure 19: Examples of the frontier of the slice problem for NDPs z^L and z^R generated by a common integer solution, having from one to four nondominated line segments.

$\vec{w}^T z(y^*)$, then $z(y^*)$ must be dominated by the slice of x_I^*, as shown in Figure 20a. Furthermore, the entire nondominated frontier within $B(z^L, z^R)$ is given by the slice for x_I^*. In this case, we solve the slice problem for x_I^* using dichotomic search (Aneja and Nair 1979), which solves a series of LPs. Therefore, if $z(y^*)$ is on or above the line segment $L(z^L, z^R)$, the SIS enhancement will generate the entire nondominated frontier within $B(z^L, z^R)$ by solving just one scalarized IP (with one no-good constraint) and a sequence of LPs. Contrast this with the basic Boxed Line Method: when there are n such NLSs in the frontier (all generated by the same integer solution), it would solve n lexicographic IPs and n scalarized IPs. Thus the SIS enhancement has the potential to provide significant savings in the number of IPs solved.

Now consider the other case, that $z(y^*)$ is below the line segment $L(z^L, z^R)$, i.e., that $\vec{w}^T z(y^*) < \vec{w}^T z^L$. There are two sub-cases: either $z(y^*)$ is an NDP, or $z(y^*)$ is dominated by some point on the (so far unknown) slice of x_I^*. These subcases are illustrated in Figures 20b and 20c, respectively.

In this, second, case, the SIS enhancement solves the LP

$$\min\{\vec{w}^T z(x) : z_1(x) \leq z_1(y^*), \quad z_2(x) \leq z_2(y^*), \quad x_I = x_I^*, \quad x \in \mathbb{X}\}. \tag{21}$$

If (21) is infeasible, then $z(y^*)$ is an NDP. This follows since $z(x) \leq z(y^*)$ for $x \in \mathbb{X}$ implies, by the definition of y^*, that either $z(x) = z(y^*)$ or $x_I = x_I^*$. The latter would imply x_I is feasible for (21), which is impossible. Hence $z(x) = z(y^*)$; $z(y^*)$ must be an NDP. Otherwise, for similar reasons, any feasible solution of (21), $\hat{x}$ say, generates an NDP $z(\hat{x})$. In either case, a new NDP has been found, and then the inner loop is called with the box $B(z^L, z^R)$ and this new NDP as z^*.

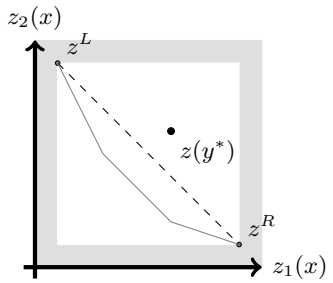
In this case, the SIS enhancement solves one scalarized IP with a no-good constraint and one LP before calling the inner loop, where the output will be a single NLS (and some number of boxes added to the queue). On the other hand, the basic Boxed Line Method would solve one lexicographic IP before calling the inner loop, with similar output. Therefore, the computational effort (and return) is comparable, and so there is not much benefit from the SIS enhancement in this case.

Computational experiments with the SIS enhancement show enough improvement to indicate that the first case is much more likely than the second case, and so, on balance, the enhancement is useful.

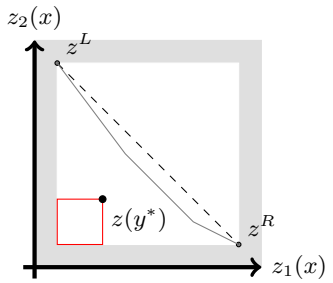
6 Implementation Issues

Because computers use finite precision arithmetic, it is necessary to introduce numerical tolerances. We use a value $\epsilon > 0$ to indicate the accuracy at which we expect the Boxed Line Method to provide output (defined more precisely in Section 6.1). Note that because the choice of ϵ impacts (the accuracy of) the nondominated frontier generated by an algorithm, it also impacts the time required by the algorithm to find the nondominated frontier. In Section 6.2, we show that the nondominated frontiers generated for the same instance for different values of ϵ can vary drastically.

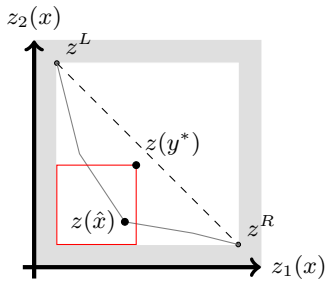
Single-objective IP solvers use tolerance as well, e.g., feasibility and optimality tolerances, and their choice also impacts the performance. In fact, certain choices of IP solver tolerance may lead to unexpected behavior in the algorithm. An illustration is provided in Figure 21. This is just one example of the



(a) If $z(y^*)$ is found on or above $L(z^L, z^R)$ (dashed), then the frontier is generated entirely by x_I^*. The slice (gray) is determined by solving LPs.

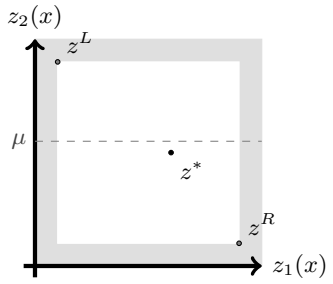


(b) If $z(y^*)$ is found below $L(z^L, z^R)$, an LP over the red region is solved. Here it determines that the slice for x_I^* does *not* dominate $z(y^*)$, and the inner loop is called with NDP $z^* = z(y^*)$.

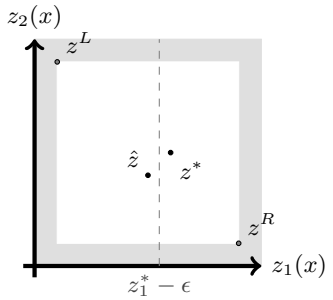


(c) In this case, the LP over the red region finds the point $z(\hat{x})$ on the slice of x_I^*, which dominates $z(y^*)$, so the inner loop is called with NDP $z^* = z(\hat{x})$.

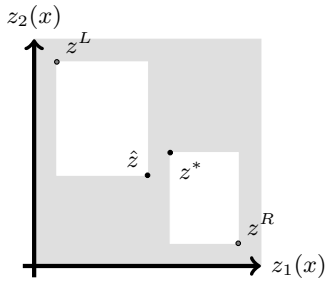
Figure 20: The SIS enhancement, applied when z^L and z^R are both generated by integer solution x_I^*. The scalarization (20) with no-good constraint $x_I \neq x_I^*$ will find a point $z(y^*)$ either above or below the line segment $L(z^L, z^R)$, proceeding with one of the three cases illustrated in (a), (b) and (c) above.



(a) In $B(z^L, z^R)$, the first lexicographic IP finds NDP z^* below the split line μ.



(b) In $B(z^L, z^R)$, the second lexicographic IP finds NDP $\hat{z}$ with $\hat{z}_1 \leq z_1^*$, but also with $\hat{z}_2 \leq z_2^* < \mu$, i.e., $\hat{z}$ dominates z^*.



(c) The two boxes in white would be added to the queue. However, a cornerpoint being dominated violates one of the most basic assumptions of our algorithms.

Figure 21: If IP solver tolerances are not set appropriately, i.e., strictly less than the algorithm's ϵ, then IPs (especially lexicographic IPs) may return points that are not nondominated.

numerical issues that we encountered during the implementation of our proposed algorithms. We found that it is critical that the IP solver tolerances are set to values strictly smaller than ϵ.

The black box nature of (commercial) IP solvers also makes it difficult to intuit how redundant criterion space constraints impact its performance. For instance, each box, $B(z^1, z^2)$, can be represented by four constraints, i.e., $z_1^1 \leq z_1(x) \leq z_1^2$ and $z_2^2 \leq z_2(x) \leq z_2^1$, or by just two constraints, i.e., $z_1(x) \leq z_1^2$ and $z_2(x) \leq z_2^1$ (the fact that z^1 and z^2 are nondominated implies the remaining constraints). Our computational experiments with CPLEX indicated that the former performs better. However, this behavior may be different with other solvers.

6.1 Epsilon Frontier

Recognizing that computers use finite precision arithmetic, we define the *ϵ-approximation of the nondominated frontier* $\mathcal{N}$, or simply the *ϵ-frontier*, $\mathcal{N}_\epsilon$, to be a set of points in criteria space such that, for fixed $\epsilon > 0$, (1) for all $z \in \mathcal{N}$, there exists $\bar{z} \in \mathcal{N}_\epsilon$ with $||z - \bar{z}||_2 \leq \epsilon$, and (2) for all $\bar{z} \in \mathcal{N}_\epsilon$, there exists

$z \in \mathcal{N}$ with $||z - \bar{z}||_2 \leq \epsilon$. Note that there may be many distinct sets that satisfy the conditions for being an ϵ-frontier. The Boxed Line Method (when run to completion) produces an ϵ-frontier for any given BOMIP having nonempty and bounded feasible set. To see how ϵ is used in the method to produce the ϵ-frontier, see the algorithms given in the Appendix.

This definition of the epsilon frontier motivated specific design choices in our implementation. For example, when the inner loop solves scalarization IPs, e.g. (7), to find NDPs that dominate a line segment, our definition implies that we are only interested in NDPs whose distance from the line segment are greater than ϵ. Therefore, we designed the criterion for entering the inner loop's while loop to be $\vec{w}^T z(y^*) < \vec{w}^T z^* - \epsilon ||\vec{w}^T||_2$ where y^* is the solution to the scalarized IP. By normalizing the gradient vector $\vec{w}$ with respect to the 2-norm at the end of the line generation subroutine, i.e., by requiring $||\vec{w}^T||_2 = 1$, we can simplify the criterion to $\vec{w}^T z(y^*) < \vec{w}^T z^* - \epsilon$.

6.2 Epsilon Sensitivity of Historical Instances

The benchmark instances used in previous computational studies on algorithms for BOMIPs are those proposed by Mavrotas and Diakoulaki (1998). The nondominated frontiers for the 20 instances, ranging in size from 20 to 320 decision variables, have a structure that was described in Section 5.2: much of the nondominated frontier consists of continuous line segments from the same slice (Figure 18). This structure not only poses numerical challenges when the line segments are extremely small, i.e., when their lengths are smaller than ϵ, it also means that changes in the value of ϵ result in drastically different frontiers being generated by our algorithms, as shown in Table 1.

Instance	10^{-3}	10^{-4}	10^{-5}	10^{-6}
1	2570	2752	2783	2786
2	2643	2916	2958	2970
3	2578	2723	2750	2753
4	5608	6121	6164	6174
5	2824	3054	3096	3096

Table 1: Number of nondominated line segments found for different values ϵ. All five instances are from the C160 class of instances and are solved by the basic method.

The fact that the NDF output by an algorithm for solving BOMIPs can be considerably different when a different tolerance value is used makes it very difficult to compare the performance of such algorithms. The issue is compounded by the fact that on algorithm, such as TSA, for example, may output several small line segments whose union is equivalent to a single line segment output by another algorithm, such as ϵTCM. This has motivated us to generate new instances for which the exact frontier is known and the line segments in the frontier all have length greater than a pre-specified value, e.g., 10^{-4}.

7 Instance Generation

We now provide a method for generating a BOMIP having a nondominated frontier controlled by parameters and known, *a priori*. In doing so, we provide an approach to "reverse engineer" a BOMIP from the feasible sets of its slice problems.

All our instances have the objective functions $z_1(x) := x_1$ and $z_2(x) := x_2$, where $x \in \mathbb{R}^2$ is a vector of two continuous decision variables. Each instance's NDF includes some sections of the line segment $L = \{(x_1, x_2) \in \mathbb{R}^2 : x_1 + x_2 = 0, -k \leq x_i \leq k \text{ for } i = 1, 2\}$ where $k \in (0, \infty)$ is a parameter. This line segment will be one slice problem in the instance. The instance has n other slice problems, where n is a parameter, all of which have their image in criterion space given by a pointed cone. The vertices of each cone lie on a line segment parallel to L but shifted vertically down by $d \in (0, \frac{2k}{n})$ units: they lie on

$$L_d = \{(x_1, x_2) : x_1 + x_2 = -d, -k \leq x_i \leq k - d \text{ for } i = 1, 2\},$$

where d is a parameter. The slice problem with vertex $(a, b) \in L_d$ has feasible set

$$P_{(a,b)}(\theta_1, \theta_2) = \{x \in \mathbb{R}^2 \ : \ \theta_1 x_1 + (1 - \theta_1) x_2 \geq \theta_1 a + (1 - \theta_1) b \tag{22}$$

$$\theta_2 x_1 + (1 - \theta_2) x_2 \geq \theta_2 a + (1 - \theta_2) b\}, \tag{23}$$

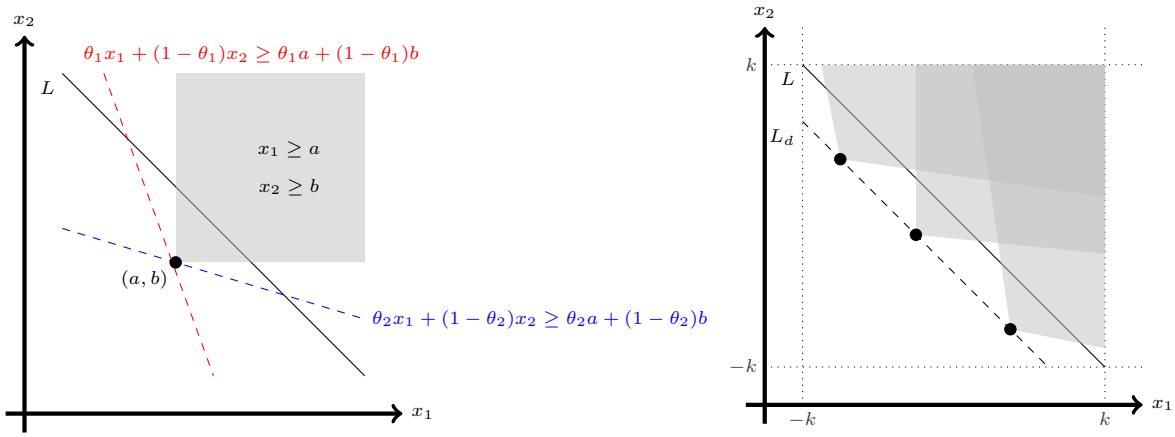


Figure 22: (Left) The orthogonal pointed cone with vertex (a, b) is shaded. The generalized pointed cone is defined between the red and blue inequalities. Different slice problems in an instance may have different values of θ_1 and θ_2. (Right) An example with 4 slice problems, having feasible sets L and the three cones.

where $\theta_1 \in [\frac{1}{2}, 1]$ and $\theta_2 \in [0, \frac{1}{2}]$ are parameters that control the width of the cone. These ranges for θ_1 and θ_2 ensure that the resulting cone contains $(a, b)+\mathbb{R}^2_+$ and is contained in the half-space that contains $L_d+\mathbb{R}^2_+$. Figure 22 illustrates this structure, showing the image of the feasible set in criterion space for 4 slice problems: L and three pointed cones with vertices lying on L_d. Note that $\theta_1 = 1$ and $\theta_2 = 0$ imply that the cone is precisely $(a, b) + \mathbb{R}^2_+$, and hence that (a, b) is an isolated NDP. Otherwise, if $\theta_1 < 1$, the left-hand boundary of the cone, from (a, b) to its intersection with L, is an NLS, while if $\theta_2 > 0$, the right-hand boundary of the cone, from (a, b) to its intersection with L, is an NLS. All instances are constructed to have the property that no two cones overlap within the band between L and L_d. This means that L alternates between a section that is part of the NDF and a section dominated by one cone.

Given any set of n polyhedra, say $P^i = \{x \in \mathbb{R}^m \ : \ A^i x \geq c^i\}$, for $i = 1, \ldots, n$, the MIP feasible set $\{(x, y) \in \mathbb{R}^m \times \{0, 1\}^n \ : \ A^i x \geq c^i - M_i(1 - y_i), \ \forall i = 1, \ldots, n, \ \sum_{i=1}^n y_i = 1\}$ has (for appropriately chosen big-M values, $(M_i)_{i=1}^n$) slice problem feasible sets $\{P^i\}_{i=1}^n$. We take $m = 2$ and use the objective function vector $z(x) = (x_1, x_2)$ so that any polyehdron in criterion space is easily reverse-engineered to give a polyhedron in decision space (they have precisely the same description).

For the $n + 1$ polyhedra consisting of L and n pointed cones $P_{(a_i,b_i)}(\theta_1^i, \theta_2^i)$, for $i = 1, \ldots, n$, where (a_i, b_i) denotes the vertex of the ith cone, this gives the following BOMIP:

$$\begin{aligned}
\text{minimize} \quad & (x_1, x_2) && && (24)\\
\text{s.t.} \quad & x_1 + x_2 \geq -2k(1 - y_0) && && (25)\\
& \theta_1^i x_1 + (1 - \theta_1^i)x_2 \geq \theta_1^i a_i + (1 - \theta_1^i)b_i - 2k(1 - y_i) && \forall i = 1, 2, ..., n && (26)\\
& \theta_2^i x_1 + (1 - \theta_2^i)x_2 \geq \theta_2^i a_i + (1 - \theta_2^i)b_i - 2k(1 - y_i) && \forall i = 1, 2, ..., n && (27)\\
& \sum_{i=0}^{n} y_i = 1 && && (28)\\
& -k \leq x_i \leq k && \forall i = 1, 2 && (29)\\
& y \in \{0, 1\}^{n+1}. && && (30)
\end{aligned}$$

This BOMIP has a number of variables and a number of constraints that is linear in n: it has $n + 3$ variables and $4 + 2n$ constraints[2].

We generate two different classes of instances having the structure described above. One class has the n cone points distributed randomly along L_d, but has $\theta_1 = 1$ and $\theta_2 = 0$ for all cones. Since this means that all cones are orthogonal pointed cones, we refer to these as *fixed cone-width instances*. An illustration of these instances is given in Figure 23. The other class has the cone points equally spaced

[2]Its number of nonzeros is also linear in n.

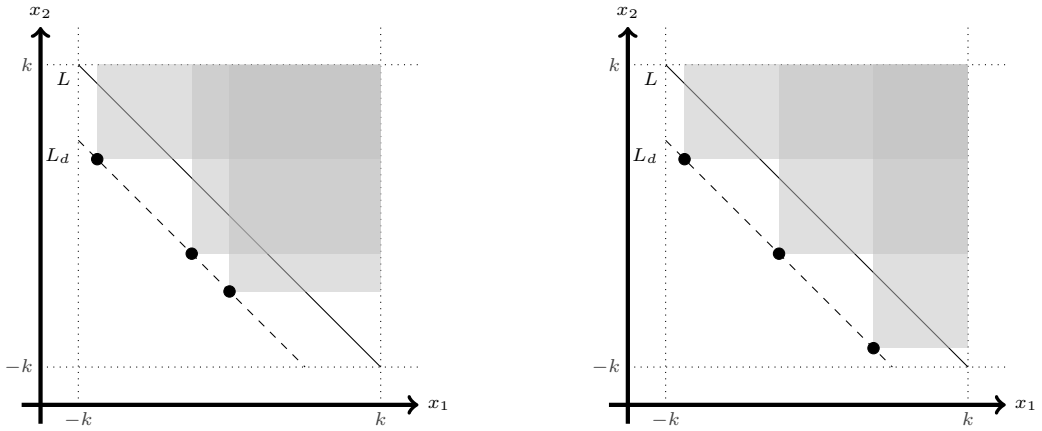


Figure 23: NDPs and associated cones generated by the fixed cone-width class of instances. (Left) If the pointed cones from NDPs overlap on L, then there are fewer than $n+1$ nondominated segments of L. (Right) Choosing the NDPs such that $-b_i < a_{i+1}$ prevents such overlapping, thus guaranteeing exactly $n+1$ line segments of L are nondominated. All instances we generate are of the latter form.

along L_d, but has randomized values of θ_1 and θ_2; we call these *randomized cone-width instances*. These are illustrated by the right-hand side of Figure 22.

All instances are carefully designed to ensure that each NLS has length at least ϵ. In the case of the fixed cone-width instances, the only NLSs are sections of L, and these can be guaranteed to have length at least ϵ by a careful choice of the randomized spacing between cone points, as a function of d. For fixed cone-width instances we take $d = k/4$, and we expect the choice of k to ensure that $d \geq \epsilon$. The latter condition also ensures that the horizontal and vertical gaps in the frontier are of length at least ϵ. In the case of random cone-width instances, an NLS may be either a section of L or a section of a cone boundary. The length of an NLS from L is ensured to be at least ϵ by restricting the values of θ_1 and θ_2 to lie within $[\frac{3}{4}, 1]$ and $[0, \frac{1}{2}]$, respectively, and choosing $d = k/(n+1) - \frac{1}{2}$. The length of an NLS generated by a cone boundary is ensured to be at least ϵ by requiring $d \geq \epsilon$. Again, we expect the choice of k to guarantee that $d \geq \epsilon$, ensuring that the horizontal and vertical gaps in the frontier are also of length at least ϵ. We force a proportion of the cones in a randomized cone-width instance to be orthogonal[3].

Specifics of our procedure for generating fixed cone-width instances are given in Algorithm 7 in the appendix. By construction, the NDF associated with such an instance has exactly n isolated NDPs and $n+1$ line segments (2 half-open at the ends of L and $n-1$ open segments in between isolated NDPs). Even large instances of this class of BOMIPs are solved quite quickly by most methods, so they are not ideal for testing computational efficiency. However, since their nondominated frontiers are known exactly, these instances are useful for validating the correctness of an algorithm.

Specifics of our procedure for generating random cone-width instances are given in Algorithm 8 and Algorithm 9 (in the appendix), which generate the cone width parameters $\{(\theta_1^i, \theta_2^i)\}_{i=1}^n$ and cone vertices, $\{(a^i, b^i)\}_{i=1}^n$, respectively. The nondominated frontiers associated with the resulting BOMIP will have no more than $3n+1$ line segments ($n+1$ from L and at most 2 per cone), including open endpoints induced by any orthogonal cones and closed endpoints induced by intersecting slices. The random cone-width class of instances is more difficult to solve, in practice, than the fixed cone-width, because of the high frequency of intersecting integer slices in the frontier.

Together, these two structured sets of instances provide a useful way to study the accuracy and robustness of a BOMIP algorithm.

8 Computational Study

This section provides computational results obtained by the Boxed Line Method (BLM) on the following sets of instances:

[3]A cone is chosen to be orthogonal with probability π, a parameter. In all our instances, we used $\pi = 0.05$.

- the ten largest instances from the benchmark instances for BOMIPs proposed by Mavrotas and Diakoulaki (1998), which we refer to as the *historical instances*, and
- ten new randomized cone-width instances, five obtained by setting $n = 5000$ and five obtained by setting $n = 7500$, which we refer to as the *new instances*.

All variations of BLM are coded in C++ and solve the linear and integer programs using IBM CPLEX Optimizer 12.6. All experiments were conducted in a single thread of a dedicated Intel Xeon ES-2630 2.3GHz with 50GB RAM, running Red Hat Enterprise Linux Server 7.4.

All variations of BLM are run with $\epsilon = 10^{-5}$ and 10^{-7} for the CPLEX tolerances for historic instances and with $\epsilon = 10^{-4}$ and 10^{-6} for the CPLEX tolerances for the new instances. The reason to use a greater accuracy for the historical instances is that very small line segments can be part of their NDF as well as nearly horizontal or vertical line segments, which makes the calculation of the gradients more critical.

We structure the discussion of the computational experiments as follows. In Section 8.1 we present a comparison of the variants of BLM, i.e., basic, multibox, no-good and recursive, and in Section 8.2, we compare (variants of) BLM with TSA and ϵTCM.

8.1 Comparison between BLM variants

Table 2 and Table 3 show the results for the historical instances and Table 4 and Table 5 show the results for the new instances. For each combination of algorithm variant and instance, we provide the following statistics: **nNDP**, the number of points output by the algorithm (described in more detail below); **nIPF**, the number of different integer part solutions (i.e., the number of different slices appearing in the NDF); **TT** the total time to discover the NDF; **IPT** the total time spent in IP solves; **LPT**, the total time spent in LP solves; **nIP**, the total number of IPs solved; **nLex**, the total number of lexicographic IPs; **nMin**, the total number of single-objective IPs solved to find the NDP that dominates an open endpoint, i.e. (10) and (11); **nScal**, the total number of scalarized IPs; **nGood**, the total number of IPs with the no-good constraint; **nLP**, the total number of LPs solved; **nRec**, the total number of boxes processed; **nSIS**, the total number of boxes processed where the corner points are generated by the same integer solution; and, finally, **nZL**, the number of boxes with z^* on the horizontal split line. When not applicable, an entry in the tables is marked with "-".

Note that when we report the number of NDPs (**nNDP**), we report the number of *distinct endpoints of line segments* in the NDF. Each (non-degenerate) line segment in the NDF, regardless of whether it is closed, half-open, or open, results in two points output by the algorithm. The algorithm outputs one point for an isolated NDP. We use **nNDP** to denote the number of points output by the algorithm, and in the remainder of this section, refer to these points as NDPs, even though in some cases, (the open ends of line segments), they are not.

The value reported in the column **nNDP** for a given instance should ideally be the same regardless of the algorithm used. However, because of the different numerical issues that a particular algorithm may encounter, they rarely produce exactly the same number, at least for the historical instances.

Note too that the total number of IPs solved (**nIP**) satisfies $nIP = 2 \times nLex + nMin + nScal + nGood$.

First, we discuss the results from Tables 2 and 3, starting with the basic variant. As shown in Figure 18, the nondominated frontier of an historical instance is characterized by continuous portions of many small nondominated line segments generated by the same integer solution. In fact, the basic variant of BLM reports an average ratio of 29.5 NDPs per integer solution (i.e., **nNDP/nIPF**) for set C160 and 47.5 NDPs per integer solution for set C320. The total time to solve a historical instance reported for the basic variant is approximately the sum of the time for solving IPs and LPs (i.e., **TT** $\approx$ **IPT** + **LPT**). The average percentage of the total time spent solving IPs is 78.1% for set C160 and 88.2% for set C320. We note that the number of lexicographic IPs is quite close to the number of NDPs. This is to be expected since in the basic variant of BLM each line segment of the NDF is discovered by first solving a lexicographic IP to obtain an NDP on this line segment. The number of scalarized IPs does not differ much from the number of NDPs as the majority of the line segments produced by the line generation subroutine can usually be proven to be part of the NDF by solving a single scalarized IP; the line segments produced by the line generation subroutine are proven to be nondominated 96.9% of the time (on average) for set C160 and 98.1% (on average) for set C320. The number of boxes processed is also quite close to the number of NDPs as each line segment of the NDF is discovered by processing exactly one box. The number of boxes with corner points with the same integer solution (**nSIS**) is large; 82.5% of the boxes (on average) for set C160 and 87.8% of the boxes (on average) for set C320. Finally, note that the vast majority of the NDPs found by solving a lexicographic IP in the outer loop fall on the horizontal split line; 99.7% of the NDPs (on average) for set C160 and 99.9% of the NDPs (on average)

for set C320. The high likelihood that the split line intersects some line segment of the NDF is a result of the fact that these instances have NDFs that contain few vertical gaps.

The results for the multibox variant are similar to those of the basic variant. The reason is that the vast majority of the line segments generated by the algorithm can be shown to be part of the NDF once the inner loop solves the first scalarized IP. In other words, the inner loop rarely executes the while-loop and therefore just a few additional boxes (those obtained from the NDPs found by solving scalarized IP (7) within the while-loop) are created.

The SIS variant, as expected, greatly improves upon the basic variant. Given a box with corner points having the same integer solution, the SIS variant is able to prove that the frontier inside the box is part of the NDF by solving a single IP with a no-good constraint most of the time. In that case, the NDF inside the box is generated by solving LPs only. The improvement of the SIS variant over the basic variant is 71.3% for total time (on average) and 79.1% for number of IP solved (on average) for set C160 and 81.2% for total time (on average) and 85.6% for number of IP solved (on average) for set C320. This is the best variant of BLM for the historical instances.

We want to draw attention to the fact that the SIS variant produces fewer NDPs for the historical instances than the other variants. We have observed that this is due to inaccuracies that can occur in the calculation of the gradients of a line segment in the line generation subroutine. If the calculated slope for a line segment is not very accurate, then the two LPs that are solved to find the endpoints z_1 and z_2 of the line segment can find incorrect endpoints. As a consequence, rather than finding the entire line segment, only a portion may be found, effectively splitting the line segment into smaller line segments. The SIS variant relies less on the line generation subroutine. When the SIS variant establishes that the frontier inside the box belongs to a single slice (i.e., belongs to the frontier associated with a single IP solution), the NDPs of that frontier are found by solving LPs without the need to calculate the individual slope of each line segment.

The recursive variant does not improve upon the basic variant because it does not recurse very often (the maximum recorded depth was 3 for both sets C160 and C320), because most of the time the full line segment obtained from the line generation subroutine is part of the NDF. Since the recursive variant rarely recurses, it proceeds similarly to the basic variant for these historical instances.

Next, we focus on the results for the new instances. The basic variant reports an average ratio of 3 NDPs per integer solution for both sets ($n = 5000$ and $n = 7500$). This is a result of their structure: since the pointed cones of the randomized cone-width instances do not overlap each other and each cone generates two lines that intersect the line segment L, and every pointed cone generates exactly 3 NDPs (even when the cone is an orthogonal cone, there is one isolated NDP and two open endpoints on L). The percentage of time spent solving IPs is 44.9% for set $n = 5000$ and 46.3% for set $n = 7500$. This is comparable to the time spent solving LPs. As expected, for the basic variant, the number of lexicographic IPs is close to the number of NDPs because there is a one-to-one correspondence between a lexicographic IP and a nondominated line segment. The number of scalarized IPs is approximately 3 per 2 NDPs. This shows that more scalarized IPs per NDP are solved in the new instances compared to the historical instances. This is due to the characterization of the nondominated frontier, where the while-loop within the inner loop must usually iterate more than once in a significant portion of the calls. Indeed, the inner loop solves a single scalarized IP 50.3% of the time (on average) for set $n = 5000$ and 59.3% of the time (on average) for set $n = 7500$. A reasonable number of the boxes processed have corner points with the same integer solution, on average of 21.8% for set $n = 5000$ and 37.4% for set $n = 7500$. This is not surprising given the way the new instances are constructed. However, *unlike* the historical instances, the frontier within these boxes is less likely to be generated entirely by that single integer solution.

The multibox variant does not improve upon the basic variant for the new instances; in fact, it is a bit slower. It produces a few more boxes (on average, the number of boxes increases by 11.13% for the set $n = 5000$ and 10.64% for the set $n = 7500$), and therefore, it may be an attractive variant for a parallel implementation. Due to the shape of the NDF, the multibox variant drastically reduces the number of boxes with corner points with the same integer solution (column **nSIS**). This is due to the fact that the NDPs discovered in the inner loop come from different pointed cones. When an NDP belonging to a cone is discovered within the while-loop of the inner loop, its LP frontier is explored in sequence until it intersects the line L. If the next iteration of the inner loop finds some other NDP, that one will necessarily belong to a different cone. Since each cone is associated with a unique integer solution, the additional rectangles obtained by the multibox variant usually have corner points from different integer solutions. The results show that the multibox variant solves slightly fewer IPs, but on the other hand solves more LPs. The result is a slight increase in the total time for solving these instances.

As expected, the SIS variant does not improve upon the basic variant for the new instances; in fact, it is a bit slower. First, in the new instances, by design, the solution of the slice problem is not as helpful

(in terms of finding multiple NDPs of the NDF associated with the integer solution of the corner points) because (1) the corner points of the box have the same integer solution of the line segment L, which is partially dominated by the pointed cones or (2) the corner points of the box have the integer solution associated to some cone (and each cone produces at most 3 NDPs). Second, we found that the solution of the slice problem (which requires solving scalarized IPs) for the new instances takes up a lot of time for the SIS variant. Those slice problems that solve scalarized IPs where the objective function has the same gradient as the line segment L may require excessive amounts of time, most likely because there are multiple optimal solutions to the scalarized IP (the NDPs on the shifted parallel line L_d). The very first scalarized IP solved may take up to 35% of the total time to find the entire NDF for an instance!

Finally, the recursive variant is very efficient for the new instances. Compared to the basic variant, the recursive variant takes, on average, only 71.9% of the total time and solves, on average, 57.2% of the number of IPs for set the $n = 5000$ and 71.5% of the total time and 55.6% of the number of IPs for set the $n = 7500$. It is more efficient because it recurses frequently with the new instances; the maximum depth level ranges from 13 to 17 for instances in the set $n = 5000$ and from 15 to 20 for instances in the set $n = 7500$. As the algorithm recurses, the number of lexicographic IPs drops significantly because many of the line segments of the NDF can be discovered and proved optimal by solving only two scalarized IPs throughout the recursion. Furthermore, since the recursive inner loop does not solve as many LPs, e.g., it does not solve (8) to find v, the total number of LPs drops noticeably compared to the basic variant.

Algorithm	Ins	nNDP	nIPF	TT	IPT	LPT	nIP	nLex	nMin	nScal	nGood	nLP	nBox	nSIS	nZL
Basic	16	2810	115	680.0	522.1	157.0	8591	2848	14	2881	-	22687	2839	2302	2830
	17	2986	110	865.4	668.1	196.4	9058	3003	7	3045	-	24920	2995	2442	2987
	18	2775	101	739.1	571.8	166.7	8399	2787	12	2813	-	22478	2778	2253	2769
	19	6204	181	2060.2	1650.6	407.2	18721	6217	18	6269	-	53756	6205	5145	6193
	20	3145	100	849.7	646.1	202.6	9596	3182	34	3198	-	26839	3168	2704	3154
Avg.		3584	121.4	1038.9	811.7	226.0	10873	3607.4	17	3641.2	-	30136	3597	2969.2	3586.6
Multibox	16	2810	115	689.9	531.2	157.9	8741	2898	14	2931	-	23057	2889	2381	2880
	17	2987	110	853.7	659.2	193.7	9238	3064	7	3103	-	25419	3056	2543	3048
	18	2775	101	722.6	559.3	162.6	8527	2830	12	2855	-	22782	2821	2324	2812
	19	6204	181	2068.4	1658.7	407.3	18957	6295	16	6351	-	54382	6285	5277	6271
	20	3145	100	894.2	680.0	212.9	9724	3225	33	3241	-	27192	3212	2771	3197
Avg.		3584.2	121.4	1045.8	817.7	226.9	11037.4	3662.4	16.4	3696.2	-	30566.4	3652.6	3059.2	3641.6
SIS	16	2764	115	206.0	148.6	57.1	1934	517	14	551	335	8287	843	335	500
	17	2952	110	257.7	185.6	71.7	1960	521	7	562	349	8915	862	349	507
	18	2738	101	232.8	170.5	61.9	1909	506	12	535	350	8172	847	350	492
	19	6166	181	577.1	433.1	143.1	3805	1018	16	1080	673	18212	1681	673	1000
	20	3122	100	216.1	148.8	66.9	1748	454	33	471	336	8698	777	336	427
Avg.		3548.4	121.4	297.9	217.3	80.1	2271.2	603.2	16.4	639.8	408.6	10456.8	1002	408.6	585.2
Recursive	16	2810	115	639.7	485.7	153.2	8212	2586	-	3040	-	22340	2577	2220	2568
	17	2987	110	835.3	639.2	195.3	8757	2799	-	3159	-	24674	2791	2416	2783
	18	2775	101	691.6	527.2	163.6	8015	2512	-	2991	-	22161	2503	2138	2494
	19	6198	181	2006.9	1594.1	410.4	18023	5718	-	6587	-	53148	5706	4964	5694
	20	3144	100	824.9	618.7	205.2	9113	2828	-	3457	-	26537	2814	2458	2800
Avg.		3582.8	121.4	999.7	773	225.5	10424	3288.6	-	3846.8	-	29772	3278.2	2839.2	3267.8

Table 2: Comparison between the different algorithms for historical instances, class C160.

Algorithm	Ins	nNDP	nIPF	TT	IPT	LPT	nIP	nLex	nMin	nScal	nGood	nLP	nBox	nSIS	nZL
Basic	21	16850	294	37665.5	33307.6	4339.7	53514	17790	40	17894	-	171370	17803	15926	17766
	22	19778	410	42045.3	36927.4	5095.7	61766	20540	23	20663	-	191611	20510	18074	20476
	23	17319	343	38995.1	34570.3	4409.4	53859	17895	26	18043	-	171514	17884	15776	17871
	24	19898	460	46967.1	41632.8	5312.7	62338	20706	34	20892	-	200619	20696	17867	20682
	25	13682	337	24450.1	21268.9	3171.0	42196	14024	27	14121	-	130934	13994	12130	13964
Avg.		17505.4	368.8	38024.6	33541.4	4465.7	54734.6	18191	30	18322.6	-	173209.6	18177.4	15954.6	18151.8
Multibox	21	16850	294	38456.5	34019.1	4419.0	53903	17920	40	18023	-	172549	17933	16204	17896
	22	19778	410	42792.9	37590.8	5180.2	62343	20732	23	20856	-	193363	20702	18447	20668
	23	17319	343	38420.5	34061.9	4345.0	54357	18062	26	18207	-	173084	18051	16080	18038
	24	19902	460	49236.4	43622.3	5592.0	62974	20920	34	21100	-	202579	20911	18243	20896
	25	13680	337	24774.0	21540.7	3223.1	42626	14169	26	14262	-	132214	14139	12416	14109
Avg.		17505.8	368.8	38736.1	34167	4551.9	55240.6	18360.6	29.8	18489.6	-	174757.8	18347.2	16278	18321.4
SIS	21	15699	294	6106.2	4890.5	1211.5	6598	1741	40	1849	1227	43443	2956	1227	1722
	22	18840	410	7788.7	6297.5	1485.3	8613	2287	23	2421	1595	52662	3850	1595	2233
	23	16449	343	6977.4	5607.1	1366.8	7459	1982	26	2131	1338	46627	3309	1338	1961
	24	18546	460	9535.6	7899.2	1630.3	10070	2672	34	2884	1808	56219	4468	1808	2679
	25	13239	337	5329.1	4296.6	1029.2	6578	1753	26	1853	1193	37911	2916	1193	1700
Avg.		16554.6	368.8	7147.4	5798.2	1344.6	7863.6	2087	29.8	2227.6	1432.2	47372.4	3499.8	1432.2	2059
Recursive	21	16831	294	37201.4	32767.3	4417.0	52297	16979	-	18339	-	170040	16971	15611	16955
	22	19763	410	41650	36417.7	5210.1	60312	19600	-	21112	-	189879	19568	17830	19536
	23	17315	343	38684.6	34149.4	4521.7	52585	17042	-	18501	-	170082	17030	15546	17018
	24	19890	460	46196.8	40725.8	5449.5	60571	19576	-	21419	-	198351	19566	17509	19552
	25	13667	337	24635	21314.5	3310.4	40899	13143	-	14613	-	129295	13113	11834	13083
Avg.		17493.2	368.8	37673.6	33074.9	4581.7	53332.8	17268	-	18796.8	-	171529.4	17249.6	15666	17228.8

Table 3: Comparison between the different algorithms for historical instances, class C320.

Algorithm	Ins	nNDP	nIPF	TT	IPT	LPT	nIP	nLex	nMin	nScal	nGood	nLP	nBox	nSIS	nZL
Basic	A	15002	5001	3996.1	1803.6	1875.2	52010	14653	398	22306	-	104487	14613	3259	14567
	B	15002	5001	3853.1	1687.7	1849.7	51738	14561	493	22123	-	103984	14519	3154	14475
	C	15002	5001	3981.6	1850.5	1819.9	51695	14544	518	22089	-	103755	14495	3193	14446
	D	15002	5001	3842.0	1699.0	1824.9	51744	14601	449	22093	-	104201	14563	3153	14521
	E	15002	5001	3948.9	1767.4	1864.2	51745	14574	474	22123	-	104228	14542	3116	14504
Avg.		15002	5001	3924.3	1761.6	1846.8	51786.4	14586.6	466.4	22146.8	-	104131	14546.4	3175	14502.6
Multibox	A	15002	5001	4022.5	1700.4	1987.3	53949	16329	339	20952	-	113616	16303	84	16221
	B	15002	5001	3925.9	1638.6	1952.6	53197	16036	430	20695	-	112354	16020	78	15943
	C	15002	5001	4040.1	1708.4	1996.3	53351	16124	428	20675	-	112264	16101	77	15989
	D	15002	5001	3909.1	1565.3	2003.1	53695	16248	368	20831	-	113210	16241	66	16140
	E	15002	5001	4068.5	1736.9	1997.6	53522	16176	387	20783	-	112744	16163	76	16057
Avg.		15002	5001	3993.2	1669.9	1987.4	53542.8	16182.6	390.4	20787.2	-	112837.6	16165.6	76.2	16070
SIS	A	15002	5001	4454.9	2278.3	1852.4	53523	16157	326	20820	63	113195	16218	63	16064
	B	15002	5001	4909.8	2623.9	1976.3	53881	16322	386	20749	102	113185	16397	102	16191
	C	15002	5001	4267.5	1969.3	1973.5	53239	16081	422	20592	63	112113	16126	63	15954
	D	15002	5001	4334.7	2025.1	1985.4	53682	16209	373	20798	93	113018	16277	93	16092
	E	15002	5001	4642.9	2328.5	1982.3	53552	16187	391	20704	83	112541	16239	83	16065
Avg.		15002	5001	4522.0	2245.0	1954.0	53575.4	16191.2	379.6	20732.6	80.8	112810.4	16251.4	80.8	16073.2
Recursive	A	15002	5001	2861.8	1383.9	1184.6	29711	3638	-	22435	-	72937	3616	1	3594
	B	15002	5001	2788.4	1301.3	1192.8	29587	3614	-	22359	-	72826	3594	1	3574
	C	15004	5001	2827.3	1350.4	1184.6	29538	3597	-	22344	-	72824	3578	1	3559
	D	15002	5001	2788.5	1306.2	1187.8	29664	3630	-	22404	-	72986	3608	1	3586
	E	15002	5001	2851.8	1367.3	1189.4	29641	3689	-	22263	-	72909	3664	1	3639
Avg.		15002.4	5001	2823.6	1341.8	1187.8	29628.2	3633.6	-	22361	-	72896.4	3612	1	3590.4

Table 4: Comparison between the different algorithms for new instances, $n = 5000$.

Algorithm	Ins	nNDP	nIPF	TT	IPT	LPT	nIP	nLex	nMin	nScal	nGood	nLP	nBox	nSIS	nZL
Basic	A	22502	7501	9268.1	4379.1	4008.0	81466	21893	669	37011	-	157416	21833	8149	21765
	B	22502	7501	9093.0	4228.6	3981.7	81333	21847	705	36934	-	157264	21796	8101	21741
	C	22502	7501	8912.3	4065.1	3984.0	81446	21847	704	37048	-	157339	21797	8192	21747
	D	22502	7501	8460.9	3818.5	3814.2	81301	21807	746	36941	-	157198	21757	8058	21705
	E	22502	7501	8672.5	4058.3	3786.0	81444	21840	708	37056	-	157230	21794	8240	21742
Avg.		22502	7501	8881.4	4109.9	3914.8	81398	21846.8	706.4	36998	-	157289.4	21795.4	8148	21740
Multibox	A	22502	7501	8642.0	3740.9	4025.8	79822	24077	593	31075	-	168439	24057	115	23962
	B	22502	7501	8726.5	3818.0	4033.7	80058	24169	598	31122	-	168754	24138	104	24035
	C	22502	7501	8774.0	3571.7	4270.0	80044	24185	592	31082	-	168694	24139	112	24023
	D	22502	7501	8686.9	3514.5	4248.5	80102	24198	620	31086	-	168730	24160	111	24048
	E	22502	7501	9054.4	3815.2	4238.6	79948	24111	627	31099	-	168691	24076	109	23988
Avg.		22502	7501	8776.8	3692.1	4163.3	79994.8	24148	606	31092.8	-	168661.6	24114	110.2	24011.2
SIS	A	22502	7501	11346.9	6662.2	3791.6	79742	24039	595	30955	114	168338	24123	114	23916
	B	22502	7501	12163.8	7132.0	4131.0	79662	23994	591	30960	123	167968	24073	123	23854
	C	22501	7501	12520.1	7754.7	3879.3	79925	24106	595	31013	105	168628	24178	105	23968
	D	22502	7501	12395.5	7320.7	4155.7	79593	23959	628	30944	103	168028	24034	103	23817
	E	22503	7501	11874.0	6862.9	4129.8	79772	24007	621	31027	110	168178	24086	110	23889
Avg.		22502	7501	12060.1	7146.5	4017.5	79738.8	24021	606	30979.8	111	168228	24098.8	111	23888.8
Recursive	A	22502	7501	6544.4	3209.7	2535.2	44466	5305	-	33856	-	109695	5278	1	5251
	B	22502	7501	6329.9	3016.2	2517.2	44485	5382	-	33721	-	109677	5345	1	5308
	C	22502	7501	6341.6	2964.6	2562.0	44445	5257	-	33931	-	109532	5228	1	5199
	D	22502	7501	6172.0	2822.3	2538.8	44415	5136	-	34143	-	109698	5099	1	5062
	E	22505	7501	6345.6	2997.1	2538.9	44451	5284	-	33883	-	109541	5252	2	5220
Avg.		22502.6	7501	6346.7	3002.0	2538.4	44452.4	5272.8	-	33906.8	-	109628.6	5240.4	1.2	5208

Table 5: Comparison between the different algorithms for new instances, $n = 7500$.

8.2 Comparison with existing algorithms

Comparing the performance of different algorithms is always challenging, but it is especially difficult for algorithms solving multiobjective mixed integer programs, given that it is non-trivial, in practice, to characterize an optimal solution, i.e., the nondominated frontier. A nondominated frontier can contain isolated points as well as open, half-open, and closed segments, and because computers employ finite-precision arithmetic, algorithms have to use tolerances to decide whether two values are equal or different. Changing the tolerance(s) used in an algorithm for finding the nondominated frontier of a multiobjectiove integer program, as we have seen in Section 6, can have a noticeable effect on the resulting nondominated frontier.

Unfortunately, the use of tolerances in algorithms for finding the nondominated frontier of a multi-objectiove integer program, also makes it more likely that the execution of an algorithm on a different hardware platform exhibits a different behavior, even to the point where it finds the nondominated frontier for an instance on one hardware platform but fails to do so for the same instance on another hardware platform.

In addition, to compare the performance of algorithms, it is preferable to run the algorithms on the same hardware platform, and, if at all possible, for an algorithm to be the only computationally intensive process running on the platform when computing times are recorded.

We were fortunate in that the developers of the Triangle Splitting Algorithm (TSA) (Boland et al. 2015b) and the ϵ,Tabu-Constraint algorithm (ϵTC) (Soylu and Yıldız 2016) both made the source code of the implementation of their algorithms available to us. Thus, a comparison of the performance of our proposed algorithms to the performance of these two algorithms could be conducted on the same hardware platform and therefore be fair.

This worked well (mostly) for the instances used in previous computational studies on algorithms for biobjective mixed integer programming, i.e., the historical instances generated according to the scheme proposed by Mavrotas and Diakoulaki (1998), but not so well for the new instances we created. The implementation of the TSA uses an "instance reader" that was customized to the historical instances, and it was not obvious how to adapt it to the new instances. The implementation of ϵTC reads the instances correctly, but terminates after the first integer program is solved.

Furthermore, the implementation of TSA that we had access to uses a *relative* tolerance and as a result only finds an approximation of the nondominated frontier. More specifically, averaged over the instances in the set C160 and C320, TSA finds 1,519 and 3,140 NDPs, respectively, whereas the SIS variant of our proposed algorithms finds, averaged over the instances in the sets C160 and C320, 3,548 and 16,555 NDPs, respectively. Therefore, TSA finds 42.8% of the NDPs found by the SIS variant of our proposed algorithm for the instances in the set C160 and only 19.0% for the instances in set C320. As a result, it is not meaningful to compare solutions times.

The implementation of ϵTC that we had access to produced almost identical nondominated frontiers, differing by only a few NDPs, but for a few historical instances, it failed to produce a nondominated frontier (it cycled). We expect that this is due to the use of a different version of CPLEX and the use of a different hardware platform.

Because ϵTC is the most recently developed algorithm for biobjective mixed integer programs, we felt it important to perform a thorough and fair comparison on all instances, new as well as historic. Hence we implemented our own version of the method, using, whenever possible, data structures and subroutines common to BLM. We validated our implementation of ϵTC by confirming that it produced a nearly identical NDF to that produced by the original implementation, in very similar computing time, on all instances for which this was possible, i.e, for which the original implementation ran on our platform.

Our overall comparison of algorithms can be found in Table 6, which reports on the performance of the SIS variant of BLM and ϵTC on the historic instances, and the recursive variant of BLM and ϵTC on the new instances.

Instance	ϵTC							BLM						
	nNDP	**nIPF**	**TT**	**IPT**	**LPT**	**nIP**	**nLP**	**nNDP**	**nIPF**	**TT**	**IPT**	**LPT**	**nIP**	**nLP**
16	2762	115	216.6	145.8	70.3	2870	10520	2764	115	206.0	148.6	57.1	1934	8287
17	2949	110	324.9	194.1	130.2	3054	16205	2952	110	257.7	185.6	71.7	1960	8915
18	2739	101	239.3	156.4	82.4	2834	11257	2738	101	232.8	170.5	61.9	1909	8172
19	6151	181	674.1	494.9	177.1	6319	23086	6166	181	577.1	433.1	143.1	3805	18212
20	3125	100	273.0	202.4	69.9	3216	9080	3122	100	216.1	148.8	66.9	1748	8698
Avg.	3545.2	121.4	345.6	238.7	106.0	3658.6	14029.6	3548.4	121.4	297.9	217.3	80.2	2271.2	10456.8
21	15636	295	8836.8	6659.8	2165.8	15923	86957	15699	294	6106.2	4890.5	1211.5	6598	43443
22	18825	410	11671.6	9055.6	2598.7	19205	98134	18840	410	7788.7	6297.5	1485.3	8613	52662
23	16420	343	10437.3	7743.2	2682.4	16752	102411	16449	343	6977.4	5607.1	1366.8	7459	46627
24	18471	457	13028.6	10187.5	2825.8	18968	101218	18546	460	9535.6	7899.2	1630.3	10070	56219
25	13216	337	7799.5	5698.3	2092.7	13518	79133	13239	337	5329.1	4296.6	1029.2	6578	37911
Avg.	16513.6	368.4	10354.8	7868.9	2473.1	16873.2	93570.6	16554.6	368.8	7147.4	5798.2	1344.6	7863.6	47372.4
5000.A	15002	5001	12152.0	11513.9	337.8	25229	24345	15002	5001	2861.8	1383.9	1184.6	29711	72937
5000.B	15002	5001	12185.2	11565.4	337.3	25281	24201	15002	5001	2788.4	1301.3	1192.8	29587	72826
5000.C	15002	5001	12218.9	11586.8	338.4	25292	24156	15004	5001	2827.3	1350.4	1184.6	29538	72824
5000.D	15002	5001	12196.0	11563.4	338.2	25256	24272	15002	5001	2788.5	1306.2	1187.8	29664	72986
5000.E	15002	5001	12029.3	11415.1	334.1	25263	24243	15002	5001	2851.8	1367.3	1189.4	29641	72909
Avg.	15002	5001	12156.3	11528.9	337.2	25264.2	24243.4	15002.4	5001	2823.6	1341.8	1187.8	29628.2	72896.4
7500.A	22502	7501	38542.3	37022.1	701.1	37875	36407	22502	7501	6544.4	3209.7	2535.2	44466	109695
7500.B	22502	7501	38528.6	37024.5	700.7	37889	36369	22502	7501	6329.9	3016.2	2517.2	44485	109677
7500.C	22502	7501	38532.5	37004.0	701.5	37890	36374	22502	7501	6341.6	2964.6	2562.0	44445	109532
7500.D	22502	7501	36466.7	34998.8	671.7	37907	36311	22502	7501	6172.0	2822.3	2538.8	44415	109698
7500.E	22502	7501	37540.8	36076.9	671.3	37893	36369	22505	7501	6345.6	2997.1	2538.9	44451	109541
Avg.	22502	7501	37922.2	36425.3	689.3	37890.8	36366.0	22502.6	7501	6346.7	3002.0	2538.4	44452.4	109628.6

Table 6: Comparison ϵTC and BLM (SIS variant for historic and recursive variant for new instances).

We observe that the SIS variant of BLM solves fewer IPs than ϵTC on the historic instances, which results in faster solution times, 297.9 vs. 345.6, respectively, averaged over the C160 instances, and 7,147.4 vs. 10,354.8, respectively, averaged over the C320 instances.

Even though the recursive variant of BLM solves more IPs than ϵTC, and far more LPs than ϵTC, on the new instances, it still ends up having significantly faster solution times, 2,823.6 vs. 12,156.3, respectively, averaged over the $n = 5000$ instances, and 6,346.7 vs. 37,9222.2, respectively, averaged over the $n = 7500$ instances.

In summary, variants of the boxed line method not only have desirable theoretical properties (in terms of the number of IPs that need to be solved to generate the NDF), but are fast in practice and outperform existing algorithms for solving BOMIPs.

Acknowledgments

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-1650044. We thank Banu Soylu for her generosity in sharing her ϵTC code with us, and for her insightful responses to our questions.

References

Y. P. Aneja and K. P. Nair. Bicriteria transportation problem. *Management Science*, 25(1):73–78, 1979.

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

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

C. A. Coello. An updated survey of ga-based multiobjective optimization techniques. *ACM Computing Surveys (CSUR)*, 32(2):109–143, 2000.

K. Deb. *Multi-objective optimization using evolutionary algorithms*, volume 16. John Wiley & Sons, 2001.

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

R. Z. Farahani, M. SteadieSeifi, and N. Asgari. Multiple criteria facility location problems: A survey. *Applied Mathematical Modelling*, 34(7):1689–1709, 2010.

X. Gandibleux. *Multiple criteria optimization: state of the art annotated bibliographic surveys*, volume 52. Springer Science & Business Media, 2006.

D. Lei. Multi-objective production scheduling: a survey. *The International Journal of Advanced Manufacturing Technology*, 43(9-10):926–938, 2009.

J. Malczewski. Gis-based multicriteria decision analysis: a survey of the literature. *International Journal of Geographical Information Science*, 20(7):703–726, 2006.

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

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

A. Rais and A. Viana. Operations research in healthcare: a survey. *International transactions in operational research*, 18(1):1–31, 2011.

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

T. Stidsen, K. A. Andersen, and B. Dammann. A Branch and Bound Algorithm for a Class of Biobjective Mixed Integer Programs. *Management Science*, 60(April):1009–1032, 2014.

T. Vincent, F. Seipp, S. Ruzika, A. Przybylski, and X. Gandibleux. Multiple objective branch and bound for mixed 0-1 linear programming: Corrections and improvements for biobjective case. *Computers and Operations Research*, 40:498–509, 2013.

D. White. A bibliography on the applications of mathematical programming multiple-objective methods. *Journal of the Operational Research Society*, pages 669–691, 1990.

A. Zhou, B.-Y. Qu, H. Li, S.-Z. Zhao, P. N. Suganthan, and Q. Zhang. Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm and Evolutionary Computation*, 1(1):32–49, 2011.

A Pseudocode

Below, we provide pseudo-code for the basic variant of BLM, for the recursive variant of BLM, and for the generation of the new instances.

Algorithm 1 BasicOuterLoop

Input: Objective functions $z_1(x), z_2(x)$ and feasible set X are *fixed and global*

Output: N the entire nondominated frontier as a set of line segments, Q any unexplored regions of the frontier space (for partial run-time)

$z^L \leftarrow \text{lexmin}\{(z_1, z_2), IP\}$ is the UL nondominated point
$z^R \leftarrow \text{lexmin}\{(z_2, z_1), IP, ifs = z^L.solution\}$ is the BR nondominated point
$N \leftarrow point(z^L) \cup point(z^R)$
if $\min_{i=1,2}\{|z_i^L - z_i^R|\} < \epsilon$ **then**
  $Q \leftarrow \emptyset$ no region to explore
  return (N, Q)
else
  $Q \leftarrow B(z^L, z^R)$ the unexplored region defined by box with corner points z^L, z^R
end if
while $Q \neq \emptyset$ **do**
  $B(z^L, z^R) \leftarrow element(Q)$
  $Q \leftarrow setminus(Q, B(z^L, z^R))$
  $\mu \in (z_2^R, z_2^L)$ arbitrary horizontal dividing line between z^L, z^R
  $z^* \leftarrow \text{lexmin}\{(z_1, z_2) : z_2(x) \leq \mu, IP, ifs = z^R.solution\}$ find NDP in lower half
  if $\mu - z_2^* > \epsilon$ **then**
    if $\min_{i=1,2}\{|z_i^* - z_i^R|\} < \epsilon$ **then**
      $N \leftarrow N \cup point(z^*)$ if epsilon-close, add to ND frontier, do not add region to queue
    else
      $Q \leftarrow Q \cup B(z^*, z^R)$ otherwise, add a new unexplored region to the queue
    end if
    $\hat{z} \leftarrow \text{lexmin}\{(z_2, z_1) : z_1(x) \leq z_1^* - \epsilon, IP, ifs = z^L.solution\}$
    if $\min_{i=1,2}\{|\hat{z}_i - z_i^L|\} < \epsilon$ **then**
      $N \leftarrow N \cup point(\hat{z})$
    else
      $Q \leftarrow Q \cup B(z^L, \hat{z})$
    end if
  else $\{|z_2^* - \mu| < \epsilon\}$
    $(z^1, z^2, z1\_open, z2\_open, M) \leftarrow$ BasicInnerLoop(z^*, z^L, z^R)
    if $\min_{i=1,2}\{|z_i^1 - z_i^2|\} < \epsilon$ **then**
      $N \leftarrow N \cup M \cup point(z^1) \cup point(z^2)$ M *only* includes NDPs that dominate z^1 or z^2
    else
      $N \leftarrow N \cup M \cup line(z^1, z^2)$
    end if

(continued on next page)

(BasicOuterLoop continued)
if $z1\_open$ **then**
  $\hat{z}^1 \leftarrow \text{argmin}\{m_2 : m \in M \cup z^L, m_2 \geq z_2^1\}$ NDP found from Inner Loop that dominates z^1
else $\{z^1$ is closed$\}$
  $\hat{z}^1 \leftarrow z^1$ closed endpoint as LR boundary of unexplored region
end if
if $\min_{i=1,2}\{|z_i^L - \hat{z}_i^1|\} < \epsilon$ **then**
  $N \leftarrow N \cup point(z^L) \cup point(\hat{z}^1)$ add to frontier
else
  $Q \leftarrow Q \cup B\left(z^L, \hat{z}^1\right)$ add to queue
end if
if $z2\_open$ **then**
  $\hat{z}^2 \leftarrow \text{argmin}\{m_1 : m \in M \cup z^R, m_1 \geq z_1^2\}$ NDP found from Inner Loop that dominates z^2
else $\{z^2$ is closed$\}$
  $\hat{z}^2 \leftarrow z^2$ closed endpoint as UL boundary of unexplored region
end if
if $\min_{i=1,2}\{|\hat{z}_i^2 - z_i^R|\} < \epsilon$ **then**
  $N \leftarrow N \cup point(\hat{z}^2) \cup point(z^R)$ add to frontier
else
  $Q \leftarrow Q \cup B\left(\hat{z}^2, z^R\right)$ add to queue
end if
end if
end while
return (N, Q)

Algorithm 2 BasicInnerLoop

Input: nondominated point $z^* = \text{lexmin}\{(z_1, z_2) : z_2(x) \leq \mu\}$ such that z^* on the split, i.e. $z_2^* = \mu$
Input: z^L, z^R upper-left and lower-right boundaries of the rectangle to explore
Output: $(z^1, z^2, z1\_open, z2\_open, M)$ the UL and BR nondominated points endpoints defining the line segment of the nondominated frontier containing z^*; markers to tell whether each endpoint is open (TRUE) or closed (FALSE); and the nondominated points that dominate open endpoints in M

$(z^1, z^2, \vec{w}, w\_known) \leftarrow LineGen(z^*, z^L, z^R)$ generate line segment of frontier that contains z^* for integer vector x_I^* via Line Generation subroutine (line segment defined by two endpoints) and return slope of the line segment to be used for scalarization
if $|z_2^1 - z_2^L| < \epsilon$ and $|z_1^1 - z_1^L| > \epsilon$ **then**
  $z1\_open \leftarrow TRUE$ because z^L dominates z^1
else
  $z1\_open \leftarrow FALSE$ temporary status with best of known information
end if
if $|z_1^2 - z_1^R| < \epsilon$ and $|z_2^2 - z_2^R| > \epsilon$ **then**
  $z2\_open \leftarrow TRUE$
else
  $z2\_open \leftarrow FALSE$
end if
$M \leftarrow \emptyset$
if $\min_{i=1,2} \left\{|z_i^1 - z_i^2|\right\} < \epsilon$ **or** $\neg w\_known$ **then**
  return $(z^*, z^*, z1\_open, z2\_open, M)$ ND segment is just the isolated nondominated point z^*
end if
$y^* \leftarrow solution \left(\min\{\vec{w}^T z(x) : z_1(x) \leq z_1^2 - \epsilon(z2\_open), \quad z_2(x) \leq z_2^1 - \epsilon(z1\_open), \quad IP, ifs = z^*.solution\}\right)$ (epsilon adjustments only for open endpoints)
while $\vec{w}^T z(y^*) < \vec{w}^T z^* - \epsilon$ (z^* is suboptimal to $z(y^*)$ w.r.t. $\vec{w}$) **do**
  if $\min_{i=1,2} \{|z_i^* - z_i(y^*)|\} < \epsilon$ and $max_{i=1,2} \{|z_i^* - z_i(y^*)|\} > \epsilon$ **then**
    ERROR message
  end if
(continued on next page)

(BasicInnerLoop continued)
if $z_1(y^*) \leq z_1^* - \epsilon$ **then**
  $\hat{v}.solution \leftarrow \min\{z_2(x) : \vec{w}^T z(x) \leq \vec{w}^T z^*, \;\; x_I = y_I^*, \;\; LP, \; ifs = y_I^*\}$ explore the slice of y_I^*
  $\hat{v} \leftarrow z(\hat{v}.solution)$
  if $\hat{v} \in line(z^1, z^2)$ **then**
    $\hat{v}.solution \leftarrow \min\{z_1(x) : z_2(x) \leq \hat{v}_2 + \epsilon, \;\; x_I = y_I^*, \;\; LP, \; ifs = y_I^*\}$ correct $\hat{v}$
    $\hat{v} \leftarrow z(\hat{v}.solution)$
  end if
  if $\hat{v} \in line(z^1, z^2)$ **then**
    $z^1 \leftarrow \hat{v}$
    $z1\_open \leftarrow FALSE$
  else $\{\hat{v} \notin line(z^1, z^2)\}$
    $v^1 \leftarrow \hat{v}$ keep track of most recent $\hat{v}$ that dominates z^1
    $z^1 \leftarrow point(line(z^1, z^2), z_2 = \hat{v}_2)$
    $z1\_open \leftarrow TRUE$
  end if
end if
if $z_1(y^*) \geq z_1^* + \epsilon$ **then**
  $\hat{v}.solution \leftarrow \min\{z_1(x) : \vec{w}^T z(x) \leq \vec{w}^T z^*, \;\; x_I = y_I^*, \;\; LP, \; ifs = y_I^*\}$ explore the slice of y_I^*
  $\hat{v} \leftarrow z(\hat{v}.solution)$
  if $\hat{v} \in line(z^1, z^2)$ **then**
    $z^2 \leftarrow \hat{v}$
    $z2\_open \leftarrow FALSE$
  else $\{\hat{v} \notin line(z^1, z^2)\}$
    $v^2 \leftarrow \hat{v}$ keep track of most recent $\hat{v}$ that dominates z^2
    $z^2 \leftarrow point(line(z^1, z^2), z_1 = \hat{v}_1)$
    $z2\_open \leftarrow TRUE$
  end if
end if
$y^* \leftarrow solution\left(\min\{\vec{w}^T z(x) : z_1(x) \leq z_1^2 - \epsilon(z2\_open), \;\; z_2(x) \leq z_2^1 - \epsilon(z1\_open), \;\; IP, \; ifs = z^*\}\right)$
end while
if $z1\_open$ and $|z_2^1 - z_2^L| > \epsilon$ **then**
  $\hat{z}^1 \leftarrow point(\min\{z_1(x) : z_2(x) \leq z_2^1, \; IP, \; ifs = v^1.solution\}, z_2^1)$ NDP that dominates z^1
  $M \leftarrow M \cup \hat{z}^1$
end if
if $z2\_open$ and $|z_1^2 - z_1^R| > \epsilon$ **then**
  $\hat{z}^2 \leftarrow point(z_1^2, \min\{z_2(x) : z_1(x) \leq z_1^2, \; IP, \; ifs = v^2.solution\})$ NDP that dominates z^2
  $M \leftarrow M \cup \hat{z}^2$
end if
return $(z^1, z^2, z1\_closed, z2\_closed, M)$

Algorithm 3 RecursiveOuterLoop

Input: Objective functions $z_1(x), z_2(x)$ and feasible set X are *fixed and global*

Output: N the entire nondominated frontier as a set of line segments, Q any unexplored regions of the frontier space (for partial run-time)

$z^L \leftarrow \text{lexmin}\{(z_1, z_2), IP\}$ is the UL nondominated point
$z^R \leftarrow \text{lexmin}\{(z_2, z_1), IP, ifs = z^L.solution\}$ is the BR nondominated point
$N \leftarrow point(z^L) \cup point(z^R)$
if $\min_{i=1,2}\{|z_i^L - z_i^R|\} < \epsilon$ **then**
  $Q \leftarrow \emptyset$ no region to explore
  return (N, Q)
else
  $Q \leftarrow B(z^L, z^R)$ the unexplored region defined by box with corner points z^L, z^R
end if
while $Q \neq \emptyset$ **do**
  $B(z^L, z^R) \leftarrow element(Q)$
  $Q \leftarrow setminus(Q, B(z^L, z^R))$
  $\mu \in (z_2^R, z_2^L)$ arbitrary horizontal dividing line between z^L, z^R
  $z^* \leftarrow \text{lexmin}\{(z_1, z_2) : z_2(x) \leq \mu, IP, ifs = z^R.solution\}$ find NDP in lower half
  if $\mu - z_2^* > \epsilon$ **then**
    if $\min_{i=1,2}\{|z_i^* - z_i^R|\} < \epsilon$ **then**
      $N \leftarrow N \cup point(z^*)$ if epsilon-close, add to ND frontier, do not add region to queue
    else
      $Q \leftarrow Q \cup B(z^*, z^R)$ otherwise, add a new unexplored region to the queue
    end if
    $\hat{z} \leftarrow \text{lexmin}\{(z_2, z_1) : z_1(x) \leq z_1^* - \epsilon, IP, ifs = z^L.solution\}$
    if $\min_{i=1,2}\{|\hat{z}_i - z_i^L|\} < \epsilon$ **then**
      $N \leftarrow N \cup point(\hat{z})$
    else
      $Q \leftarrow Q \cup B(z^L, \hat{z})$
    end if
  else $\{|z_2^* - \mu| < \epsilon\}$
    $L \leftarrow \emptyset$ no known line segments so far
    $M \leftarrow$ RecursiveInnerLoop(z^*, z^L, z^R, L) M includes all found ND points and line segments

(continued on next page)

(RecursiveOuterLoop continued)
$M_< \leftarrow Orderbyz1(\{z^L, z^R\} \cup M)$ giving ordered set $(z^L, m^1, ..., m^k, z^R)$
for $\forall$ consecutive pairs $(a, b) \subset M_<$ **do**
  if $line(a, b) \in M$ **then**
    $N \leftarrow N \cup line(a, b)$ add line segment to frontier
  else
    $N \leftarrow N \cup point(a) \cup point(b)$ add individual points to frontier
    if $\min_{i=1,2}\{|a_i - b_i|\} > \epsilon$ **then**
      $Q \leftarrow Q \cup B\,(a, b)$ add unexplored region to queue
    end if
  end if
end for
end if
end while
return (N, Q)

Algorithm 4 Inner Loop: Recursive

Input: nondominated point z^*
Input: z^L, z^R known NDPs providing the upper-left and lower-right corner points of the box
Input: L the set of all inherited line segments so far (for line trimming); note that $L = \emptyset$ on first call
Output: M all found nondominated points/line segments

$(z^1, z^2, \vec{w}, w\_known) \leftarrow LineGen(z^*, z^L, z^R)$
if w_known **then**
  $(z^1, z^2) \leftarrow LineTrim(z^*, z^1, z^2, \vec{w}, L)$
end if
if $|z_2^1 - z_2^L| < \epsilon$ and $|z_1^1 - z_1^L| > \epsilon$ **then**
  $z1\_open \leftarrow TRUE$ because z^L dominates z^1
else
  $z1\_open \leftarrow FALSE$ temporary status with best of known information
end if
if $|z_1^2 - z_1^R| < \epsilon$ and $|z_2^2 - z_2^R| > \epsilon$ **then**
  $z2\_open \leftarrow TRUE$
else
  $z2\_open \leftarrow FALSE$
end if
$M \leftarrow \emptyset$
if $\min_{i=1,2} \left\{|z_i^1 - z_i^2|\right\} < \epsilon$ or $\neg w\_known$ **then**
  $M \leftarrow M \cup point(z^*)$ frontier is just the isolated nondominated point z^*
  return M
else
  $L \leftarrow L \cup line(z^1, z^2)$
end if
$y^* \leftarrow solution\left(\min\{\vec{w}^T z(x) : z_1(x) \leq z_1^2 - \epsilon(z2\_open),\ \ z_2(x) \leq z_2^1 - \epsilon(z1\_open),\ \ IP,\ ifs = z^*.solution\}\right)$ (epsilon adjustment only for open endpoints)
while $\vec{w}^T z(y^*) < \vec{w}^T z^* - \epsilon$ (z^* is suboptimal to $z(y^*)$ w.r.t. $\vec{w}$) **do**
  if $\min_{i=1,2} \{|z_i^* - z_i(y^*)|\} < \epsilon$ and $max_{i=1,2} \{|z_i^* - z_i(y^*)|\} > \epsilon$ **then**
    ERROR message
  end if
  $M \leftarrow M \cup z(y^*)$

(continued on next page)

(RecursiveInnerLoop continued)
if $z_1(y^*) \leq z_1^* - \epsilon$ **then**
  $\alpha \leftarrow \text{argmin}\{p_2 : p \in z^L \cup M, p_1 < z_1^*\}$ bound by the nearest (on left) found NDP to z^*
  $M' \leftarrow InnerLoop(z(y^*), \alpha, z^*, L)$ recurse within $B(\alpha, z^*)$ with inherited line segments in L
  $M \leftarrow M \cup M'$
  $L \leftarrow L \setminus line(z^1, z^2)$ remove "old" line segment
  $\hat{p} \leftarrow \text{argmin}\{p_2 : p \in M, p_1 < z_1^*\}$ choose the nearest (on left) found NDP to z^*
  if $\hat{p} \in line(z^1, z^2)$ **then**
    $z^1 \leftarrow \hat{p}$
    $z1\_open \leftarrow FALSE$
  else $\{\hat{p} \notin line(z^1, z^2)\}$
    $z^1 \leftarrow point(line(z^1, z^2), z_2 = \hat{p}_2)$
    $z1\_open \leftarrow TRUE$
  end if
end if
if $z_1(y^*) \geq z_1^* + \epsilon$ **then**
  $\beta \leftarrow \text{argmin}\{p_1 : p \in z^R \cup M, p_2 < z_2^*\}$
  $M' \leftarrow InnerLoop(z(y^*), z^*, \beta, L)$ recurse within $B(z^*, \beta)$
  $M \leftarrow M \cup M'$
  $\hat{p} \leftarrow \text{argmin}\{p_1 : p \in M, p_2 < z_2^*\}$ choose the nearest (on right) found NDP to z^*
  if $\hat{p} \in line(z^1, z^2)$ **then**
    $z^2 \leftarrow \hat{p}$
    $z2\_open \leftarrow FALSE$
  else $\{\hat{p} \notin line(z^1, z^2)\}$
    $z^2 \leftarrow point(line(z^1, z^2), z_1 = \hat{p}_1)$
    $z2\_open \leftarrow TRUE$
  end if
end if
$L \leftarrow L \cup line(z^1, z^2)$ add updated line segment
$y^* \leftarrow solution\left(\min\{\vec{w}^T z : z_1(x) \leq z_1^2 - \epsilon(z2\_open), \quad z_2(x) \leq z_2^1 - \epsilon(z1\_open), \quad IP, \; ifs = z^*.solution\}\right)$ (epsilon adjustment only for open endpoints)
end while
$M \leftarrow M \cup line(z^1, z^2)$
return M

Algorithm 5 Line Segment Generation

Input: z^* nondominated point whose integer vector is fixed
Input: z^L, z^R upper-left and lower-right boundaries of the rectangle to explore
Output: $(z^1, z^2, \vec{w}, w\_known)$ the UL and BR endpoints for the line segment of frontier including z^* and the slope of the line segment (if found)

$z1\_known, z2\_known \leftarrow FALSE$
$w\_known \leftarrow FALSE$ weight vector for which z^* is optimal (if found, can be used to directly compute z^1, z^2)
$\delta_1, \delta_2 \leftarrow 100\epsilon$ starting horizontal/vertical distance
(Explore for z^2 in lower-right:)
$i \leftarrow 0$
$t^i \leftarrow \text{lexmin}\{(z_2, z_1) : z_1(x) \leq z_1^* + \delta_1,\ LP,\ ifs = z^*.solution\}$ is the BR nondominated point
while $w\_known = FALSE$ **and** $z2\_known = FALSE$ **do**
  if $\min\left(|z_1^* - t_1^i|, |z_2^* - t_2^i|\right) < \epsilon$ **then**
    $z^2 \leftarrow z^*$
    $z2\_known \leftarrow TRUE$
    BREAK
  end if
  $\vec{w}^i \leftarrow slope(z^*, t^i)$
  $t^{i+1} \leftarrow \min\{(\vec{w}^i)^T z(x),\ LP,\ ifs = z^*.solution\}$ (scalarized by $\vec{w}^i$)
  if $(\vec{w}^i)^T t^{i+1} = (\vec{w}^i)^T z^*$ and $|z_1^* - t_1^i| > \epsilon$ and $|z_2^* - t_2^i| > \epsilon$ **then**
    $\vec{w} \leftarrow \vec{w}^i$
    $w\_known \leftarrow TRUE$
    if $i \geq 1$ and $t^i \in B(z^L, z^R)$ **then**
      $z^2 \leftarrow t^i$
      $z2\_known \leftarrow TRUE$
    end if
    BREAK
  end if
  $i \leftarrow i + 1$
end while

(continued on next page)

(Line Generation)
if $w\_known = FALSE$ **then**
  (Explore for z^1 in upper-left:)
  $i \leftarrow 0$
  $t^i \leftarrow \text{lexmin}\{(z_1, z_2) : z_2(x) \leq z_2^* + \delta_2,\ LP,\ ifs = z^*.solution\}$ is the UL nondominated point
end if
while $w\_known = FALSE$ **and** $z1\_known = FALSE$ **do**
  if $\min\left(|z_1^* - t_1^i|, |z_2^* - t_2^i|\right) < \epsilon$ **then**
    $z^1 \leftarrow z^*$
    $z1\_known \leftarrow TRUE$
    BREAK
  end if
  $\vec{w}^i \leftarrow slope(z^*, t^i)$
  $t^{i+1} \leftarrow \min\{(\vec{w}^i)^T z(x),\ LP,\ ifs = z^*.solution\}$
  if $(\vec{w}^i)^T t^{i+1} = (\vec{w}^i)^T z^*$ **then**
    $\vec{w} \leftarrow \vec{w}^i$
    $w\_known \leftarrow TRUE$
    if $i \geq 1$ **then**
      $z^1 \leftarrow t^i$
      $z1\_known \leftarrow TRUE$
    end if
    BREAK
  end if
  $i \leftarrow i + 1$
end while
(By the time the code has reached this point, either $\vec{w}$ is known OR *both* z^1 *and* z^2 are known)
if $z1\_known = FALSE$ **then**
  $z^1 \leftarrow \min\{z_1(x) : \vec{w}^T z(x) \leq \vec{w} z^* + \epsilon, z_2(x) \leq z_2^L,\ LP,\ ifs = z^*.solution\}$
end if
if $z2\_known = FALSE$ **then**
  $z^2 \leftarrow \min\{z_2(x) : \vec{w}^T z(x) \leq \vec{w} z^* + \epsilon, z_1(x) \leq z_1^R,\ LP,\ ifs = z^*.solution\}$
end if
if $w\_known = FALSE$ **then**
  $\vec{w} \leftarrow [-1, -1]$ clearly impossible gradient vector
end if
$\vec{w} \leftarrow \vec{w}/||\vec{w}||_1$
return $(z^1, z^2, \vec{w}, w\_known)$

Algorithm 6 LineTrim: Segment Trimming

Input: z^* nondominated point
Input: $z^1, z^2, \vec{w}$ upper-left and lower-right points of the line segment and the gradient vector
Input: $L = \{L_1, L_2, ..., L_n\}$ finite (possibly empty) set of line segments to test for intersection, where $L_i = L(a^i, b^i)$
Output: $(z^1, z^2, \vec{w}, w\_known)$ the endpoints for the line segment of frontier including z^* and the gradient vector (if found)

if $L = \emptyset$ **then**
  return (z^1, z^2)
end if
for i=1,2,...,n **do**
  $\vec{w}^i \leftarrow (b_1^i - a_1^i, a_2^i - b_2^i)$
  $\vec{w}^i \leftarrow \vec{w}/||\vec{w}||_1$
  if $\vec{w} \neq \vec{w}^i$ **then**
    solve SLE for γ:

$$\begin{bmatrix} \vec{w}_1 & \vec{w}_2 \\ \vec{w}_1^i & \vec{w}_2^i \end{bmatrix} \begin{bmatrix} \gamma_1 \\ \gamma_2 \end{bmatrix} = \begin{bmatrix} \vec{w}_1 z_1^* + \vec{w}_2 z_2^* \\ \vec{w}_1^i a_1^i + \vec{w}_2^i a_2^i \end{bmatrix}$$

    if $\gamma_1 \in [z_1^1, z_1^2] \cap [a_1^i, b_1^i]$ **and** $\gamma_2 \in [z_2^2, z_2^1] \cap [b_2^i, a_2^i]$ **then**
      if $\gamma_1 < z_1^*$ **then**
        $z^1 \leftarrow \gamma$
      else $\{\gamma_1 > z_1^*\}$
        $z^2 \leftarrow \gamma$
      end if
    end if
  end if
end for
return (z^1, z^2)

Algorithm 7 Fixed Cone-Width NDP Generation

$d = k/4$ (distance by which L_d is shifted down)
$w = (2k - d)/n$ (the width of subintervals)
$a_1 = U(-k, -k + w)$
$b_1 = -a_1 - d$
for $i = 2, 3, ..., n$ **do**
    $a_i = U(\max\{-k + (i-1)w, -b_{i-1} + \epsilon\}, -k + iw)$
    $b_i = -a_i - d$
end for

Algorithm 8 Randomized Theta Generation

thetalist= $\emptyset$
for $i = 1, 2, ..., n$ **do**
    if $U(0, 1) \leq \pi$ **then**
        $\theta_1 = 1$
        $\theta_2 = 0$
    else
        $\theta_1 = U(\frac{3}{4}, 1)$
        $\theta_2 = U(0, \frac{1}{4})$
    end if
    thetalist.append((θ_1, θ_2))
end for

Algorithm 9 Randomized Cone-Width NDP Generation

$d = k/(n + 1) - 0.5$
$a_1 = -k + 0.5d$
$b_1 = -a_1 - d$
for $i = 2, ..., n$ **do**
    $a_i = a_{i-1} + 2d + 1$
    $b_i = -a_i - d$
end for